www.clubedelphi.net

Feita para desenvolvedores

ClubeDelphi

DevMedia GROUP

Ano 8 · Edição 92 · R$9,90

ISSN 1517990-7

Agora com + páginas e com duas grandes novidades!

- Artigos de PHP
- e muito mais artigos para iniciantes em Delphi

Desenvolvimento para PALM

Na Web »

Confira no portal ClubeDelphi duas vídeo-aulas sobre PocketStudio

Novidades
Desenvolvimento Web RAD com VCL for the Web e AJAX no Delphi 2007

Delphi Language
Desvende e utilize na prática o poderoso recurso de Diretivas de Compilação

Relatórios
Dicas de QuickReport

Easy Delphi
Caixas de diálogo

Easy Delphi
Treinamento em ASP.NET – Parte 3

PHP
Mini-Curso - Introdução à linguagem PHP – Parte 4

PHP
Mini-curso de Delphi for PHP – Crie uma aplicação Web completa – Parte 5

Coluna: Ask the Expert

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®
The Foundation of Information Security

Alameda Tocantins, 280 - Barueri-SP
Tel: +55 11 4208-7700
http://br.safenet-inc.com - Contato: infobrasil@safenet-inc.com

ClubeDelphi

Ano 8 - 92ª Edição - 2008 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Adriano Santos
adrianosantos@devmedia.com.br

Equipe Editorial
Fabricio Desbessel, Maikel Scheid, Paulo Quicoli, Luciano Pimenta

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
antonioxavier@devmedia.com.br

Revisão
Gregory Monteiro
gregory@clubedelphi.net

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 2215-0033

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2215-0033

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

ClubeDelphi PLUS

DevMedia GROUP

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS. Para mais informações sobre o pacote PLUS, acesse: *http://www.devmedia.com.br/clubedelphi/portal.asp*

Apoio

ASSESPRO PARANÁ

EDITORIAL

Uma atualização do Delphi 2005 introduziu um compilador que permitia o suporte ao desenvolvimento para o .NET Compact Framework. Esse suporte ainda permanece no Delphi 2006/2007, e você pode conferir nas edições 89 e 90 como desenvolver para CF com Delphi. Mas obviamente quando falamos em .NET e CF estamos falando em desenvolvimento para Windows Mobile/CE e Pocket PC's. Porém, um outro tipo de dispositivo móvel amplamente utilizado no mercado é o Palm, que roda um sistema operacional diferente e incompatível com as aplicações construídas em Delphi. A saída é utilizar o Pocket Studio, excelente ferramenta que tem a "cara" do Delphi e utiliza Pascal para permitir o desenvolvimento de aplicações para esse tipo de dispositivo. Nesta edição, iniciamos um mini-curso de Ricardo Boaro, em quatro partes, que vai mostrar todos os segredos do desenvolvimento para PalmOS.

O IntraWeb, agora com o nome de VCL for the Web, é um framework que permite que você utilize os mesmos conceitos do desenvolvimento Desktop para criar aplicações para Web. Ele veio no Delphi 7, apesar de existir desde o Delphi 2, criado originalmente por Chad Hover da empresa Atozed. No Delphi 2007, o IntraWeb suporta agora o AJAX, de uma forma totalmente transparente. Você pode simplesmente programar nos eventos corretos e o framework cuida do resto: nada de aprender JavaScript, Ajax puro etc. Confira o artigo de Adriano dos Santos.

Ricardo Boaro fala de um interessante recurso da Delphi Language (antigo Object Pascal), conhecido como diretivas de compilação. Esse recurso permite que você instrua o compilador do Delphi a processar determinados blocos de código de acordo com uma condição. Ideal para resolver problemas onde nenhum outro recurso da linguagem resolve.

Mesmo com o Rave Reports, Report Builder, Fast Reports, Fortes Reports, acredito que o gerador de relatórios mais utilizado ainda pelos desenvolvedores Delphi é o Quick Report. Já trabalhei muito com ele, e posso afirmar com certeza: dá para fazer muita coisa com ele, algumas você precisa usar mais a cabeça, driblar aqui ou ali, mas dá. Então por que não falar de QR se é o gerador mais popular? E se você nunca fez relatórios com Delphi e gostaria de aprender? Confira o artigo do Maikel.

Três artigo na nossa sessão Easy Delphi deste mês, para aqueles que estão iniciando com o Delphi: aprenda tudo sobre caixas de diálogo, saiba como usar o ActionList (um dos meus componentes preferidos da VCL depois do CDS) e confira também a última parte do treinamento em ASP.NET. Já na sessão PHP, finalizamos os dois mini-cursos, inclusive com o Fabrício mostrando como hospedar uma aplicação Delphi PHP em um servidor real.

Um ótimo 2008, muito Delphi na cabeça, sucesso e boa leitura!

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@devmedia.com.br

SEÇÃO DELPHI: Artigos intermediários sobre Delphi Win32 e Delphi .net.

12 - Desenvolvendo uma Aplicação Completa com PocketStudio
Ricardo C. Boaro

20 - VCL for the Web e Ajax
Adriano Santos

26 - Diretivas de Compilação
Ricardo C. Boaro

SEÇÃO EASY DELPHI: Artigos para iniciantes em Delphi

32 - Dicas de QuickReport
Maikel Marcelo Scheid

38 - Mensagens e caixas de diálogo
Adriano Santos

46 - Treinamento em ASP.NET – Parte 3
Renato Haddad

SEÇÃO PHP: Artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

54 - Delphi for PHP - Parte 5
Fabrício Desbessel

60 - Introdução ao PHP – Parte 4
Ewald Geschwinde e Hans-Juergen Schoenig

Portal do Assinante

NÃO PERCA

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:
- Mais de 460 Vídeo Aulas!
- 6 cursos online!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 150 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse *www.devmedia.com.br/clubedelphi/potal.asp* e utilize as informações abaixo: **Login: DVM.PL e Senha: VDM19**

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Adriano Santos**
(envie as suas para falecom@adrianosantos.pro.br)

Como faço para que minha aplicação disque um número de telefone?

Gostaria de saber se é possível criar um discador via Delphi já que tenho uma aplicação capaz de guardar números de telefone.

Pedro Gomes

Olá Pedro, respondendo a sua questão: sim, existe como criar uma aplicação discadora. Crie um formulário semelhante a **Figura 1**. No ComboBox a direita inclua as strings COM1, COM2, COM3 e COM4 na propriedade Items do objeto. No código do botão *Discar* digite o código da **Listagem 1**. Veja o sistema em funcionamento na **Figura 2**.

Listagem 1. Código do Botão Discar

```
...
var
  Form1: TForm1;
  hCommFile: THandle;
  Status: LongBool;
  NumberWritten: DWORD;
  Buf: array[0..1023] of Byte;
  S: string;
...
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (ComboBox1.text <> ' ') and (Edit1.Text <> ' ') then
  begin
    S := Combobox1.Text;
    hCommFile :=
      CreateFile(PChar(s), GENERIC_WRITE, 0,
      nil,
      OPEN_EXISTING,
      FILE_ATTRIBUTE_NORMAL,
      0);
    Memo1.Lines.clear;
    Memo1.Lines.add('Discando...');
    if RadioButton1.CheCked then
      S := 'ATDT'
    else
      S := 'ATDP';
    S := S + Edit1.Text + #13#10;
    Memo1.Lines.Add(S);
    NumberWritten := 0;
    Status := WriteFile(hCommFile, PChar(s)[0],
      Length(s),
      NumberWritten,
      nil);
    Memo1.Lines.Add('Aguardando Atendimento ...');
    MessageDlg('Retire o telefone do gancho e clique OK para desligar o modem',
      mtInformation, [mbok], 0);
    WriteFile(hCommFile, 'ATH', 5, NumberWritten, nil);
    CloseHandle(hCommFile);
    Memo1.Lines.Add('Modem Desconectado.'#13#10);
  end;
end;
```

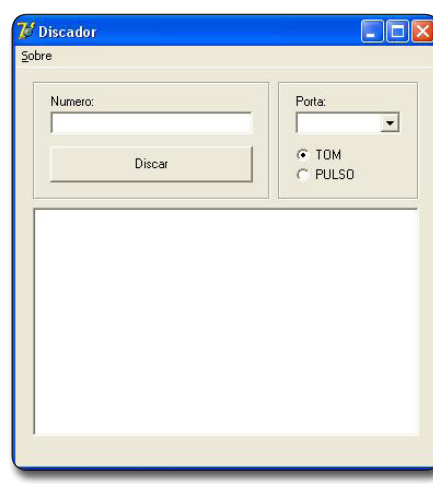


Figura 1. Layout da tela

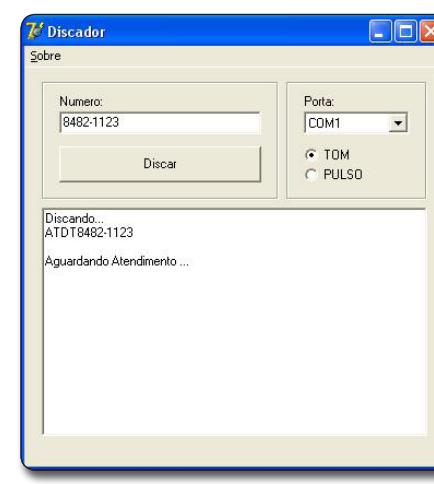


Figura 2. Sistema em execução

Como criar animações para minhas janelas e caixas de diálogo?

Olá, existem diversos programas que, ao fechar ou minimizar, executam animações na janela. Gostaria de criar algo parecido nos meus sistemas, mas não sei nem por onde começar.

Felipe Madeira

Olá Felipe, o recurso que procura já é previsto no próprio sistema operacional Windows e pode ser facilmente implementado no sistema através de chamadas as *API's* ou criação de classes. Vejamos as duas maneiras de se fazer isso.

A primeira delas consiste em criar uma animação para enviar a aplicação para o *TrayIcon*, ou seja, a famosa mandeja do Windows. Para isso criaremos uma classe simples e a usaremos em nossa aplicação.

Crie uma nova aplicação no Delphi 7.0 usando *File|New>Application* e salve o formulário principal como "uPrincipal.pas" usando o menu *File>Save as*. Coloque o nome do formulário principal como "frmPrincipal". Agora salve o projeto como "Animacoes.dpr". Como mencionado criaremos uma classe para criar nossa animação. Para isso crie uma nova *Unit* no menu *File|New>Unit*. Salve-a como "uTrayZoom.pas".

Após a palavra reservada *Interface* declare uma seção *Uses* e nela inclua a *Unit Windows*. Logo em seguida inclua o código da **Listagem 2** que possui a declaração das variáveis, funções e procedimentos de nossa classe. As classes serão abstratas, desta forma não será necessário instanciá-las.

Pressione *Ctrl + Shift + C* para que o Delphi declare pra nós o cabeçalho das *procedures* e *functions*. Codifique cada função e procedimento digitando os códigos da **Listagem 3** observando atentamente o código de cada um dos procedimentos.

Agora retorne ao formulário principal e vamos implementar as chamadas às funções.

Inclua dois *Button's* na tela com os *Caption's* "Minimizar para a bandeja" e "Maximizar a aplicação". Vamos fazer apenas a chamada às funções de animação. O esquema de envio do software para o *TrayIcon* ficará a seu critério. No *Button* "Minimizar para a bandeja" inclua o código que segue:

```
TTrayZoom.ZoomToTray(Handle);
```

E no botão "Maximizar a aplicação" inclua o seguinte código:

```
TTrayZoom.ZoomFromTray(Handle);
```

> **Nota**: Não esqueça de adicionar a *Unit uTrayZoom* ao *Uses* do formulário principal usando *File>Use unit*.

Note que não estamos instanciando a classe *TrayZoom*. Isso porque criamos seus métodos como abstratos. O layout da tela deverá se parecer com a **Figura 2**. Por fim rode a aplicação e testes os dois botões. Perceba que acontece uma pequena animação da janela ao, supostamente, minimizar ou maximizar da bandeja do Windows.

Agora faremos uma nova alteração, desta vez utilizando a *API* do Windows *DrawAnimatedRects*. Para isso crie uma nova janela no aplicativo usando *File|New>Form*. Salve-o como "uDialogo.pas" e seu *Name* para "DialogoForm". Pressione *F12* no Delphi para visualizar o código do formulário. Inclua uma nova propriedade como segue:

```
property SourceCtrl: TControl
read fSourceCtrl write fSourceCtrl;
```

Na área *private* crie uma variável *TControl*, uma *procedure* e uma *function* conforme a seguir:

```
...
private
  fSourceCtrl: TControl;
  function GetSourceCtrlBounds: TRect;
  procedure AnimateDialog(const Src, Dest: TRect);
...
```

No evento *OnHide* do formulário codifique como a seguir:

```
if WindowState <> wsMinimized then
  AnimateDialog(BoundsRect,
  GetSourceCtrlBounds);
```

E no evento *OnShow* digite o código que segue:

```
if WindowState <> wsMinimized then
  AnimateDialog(GetSourceCtrlBounds,
  BoundsRect);
```

Em seguida basta codificar o restante dos procedimentos (*GetSourceCtrlBounds* e *AnimateDialog*) conforme a **Listagem 3**. Retorne ao formulário principal e inclua a *Unit* da nossa caixa de diálogo ao *Uses* da tela principal. Agora inclua um novo botão com o *Caption* "Diálogo". Codifique-o com o código a seguir:

```
DialogForm.Visible := not DialogForm.Visible;
```

Rode novamente a aplicação e teste-a. Perceba que agora acontece uma animação ao abrir a caixa de diálogo. Enfim, o processo é simples. Basta um pouco de criatividade e tudo se resolve.

Um forte abraço ●

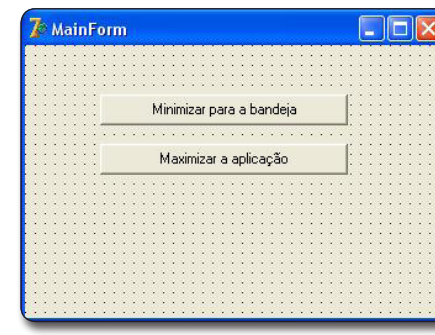


Figura 2. Exemplo de tela

Listagem 2. Declaração das variáveis, funções e procedimentos

```
unit uTrayZoom;

interface
uses
  Windows;

type
  TTrayZoom = class(TObject)
  private
    class function GetTrayRect: TRect;
    class procedure DoZoom(const Wnd: HWND; const Src, Dest: TRect);
  public
    class procedure ZoomToTray(const Wnd: HWND);
    class procedure ZoomFromTray(const Wnd: HWND);
  end;
```

Listagem 3. Código de cada procedimento

```
class procedure TTrayZoom.DoZoom(const Wnd: HWND; const
  Src, Dest: TRect);
begin
  DrawAnimatedRects(Wnd, IDANI_CAPTION, Src, Dest);
end;

class function TTrayZoom.GetTrayRect: TRect;
var
  TaskbarWnd, TrayWnd: HWND;
begin
  TaskbarWnd := FindWindow('Shell_TrayWnd', nil);
  TrayWnd := FindWindowEx(TaskbarWnd, 0, 'TrayNotifyWnd',
    nil);
  GetWindowRect(TrayWnd, Result);
end;

class procedure TTrayZoom.ZoomFromTray(const Wnd: HWND);
var
  WndRect: TRect;
begin
  GetWindowRect(Wnd, WndRect);
  DoZoom(Wnd, GetTrayRect, WndRect);
end;

class procedure TTrayZoom.ZoomToTray(const Wnd: HWND);
var
  WndRect: TRect;
begin
  GetWindowRect(Wnd, WndRect);
  DoZoom(Wnd, WndRect, GetTrayRect);
end;
```

Assinatura
ClubeDelphi PLUS
Mais conteúdo .NET por menos!

+460 vídeo aulas | 6 cursos online

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 460 vídeo aulas; ii) 6 cursos online; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 150 artigos exclusivos (que não foram publicados na revista)!;

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia

.:: DESTAQUE ::.

Confira no portal ClubeDelphi PLUS 2 vídeo aulas sobre PocketStudio

PocketStudio e PalmOS
http://www.devmedia.com.br/articles/viewcomp.asp?comp=6443
http://www.devmedia.com.br/articles/viewcomp.asp?comp=7349

Curso Online - PLUS

O portal ClubeDelphi PLUS conta com 6 cursos online, confira!

Sistema completo com Delphi 7, dbExpress e Firebird 2.0 [*Luciano Pimenta*]

Aplicações WEB com IW e Delphi 7 (Delphi Win32)! [*Guinther Pauli*]

Aplicações client/server no Delphi 2006 [*Luciano Pimenta*]

Criando uma Aplicação multi-camadas Completa com Delphi [*Guinther Pauli*]

Criando uma aplicação completa: sistema SysCash [*Everson Volaco*]

Aplicações Client/Server com dbExpress e Firebird [*Luciano Pimenta*]

Últimas Vídeo-Aulas (Somente para assinantes)

Desenvolvendo uma aplicação para PalmOS com PocketStucio - Parte VII a IX
Veja nessas vídeo aulas de Ricardo Boaro, como trabalhar com aplicações Palm/OS com o PocketStudio.

Iniciantes no ActionList
Veja nesta vídeo aula como trabalhar com este poderoso componente da VCL do Delphi. Com ele podemos agrupar as várias ações do sistema.

Acessando SQL Server com dbGo no Delphi 2007
Veja nessa vídeo aula de Ricardo Boaro as principais características do componente AdoTable.

Crie uma loja virtual com Delphi for PHP – Parte IX
Acompanhe a continuação do curso de Delphi for PHP do Fabrício Desbessel.

Rave Reports GlobalPages, utilizando section e DataMirrors – Parte I a VII
Veja nessa vídeo aula de Jefferson Junglaus um mini-curso de Rave Reports com DataMirrors e GlobalPages.

Rave Reports – Alterando o visualizador de relatórios do Rave
Veja nessa vídeo aula de Jefferson Junglaus, como alterar o visualizador Rave Reports no Delphi.

Aprenda a desenvolver sistemas para o sistema operacional PalmOS
Acompanhe as aulas de Ricardo Boaro que falam unicamente do desenvolvimento de aplicações para PalmOS utilizando a IDE PocketStudio que é bastante semelhate ao Delphi inclusive utilizando-se de linguagem Pascal.

Errata

Código Fonético no Firebird - Edição 82

O artigo "Código fonético no Firebird", edição 82, do autor Jorge Luis Bublitz, mostra de maneira prática como implementar uma rotina para busca de nomes em um banco de dados Firebird utilizando-se de fonética, ou seja, a forma como o nome é pronunciado. O autor revela que há basicamente duas implementações capazes de tal implementação: Soundex e Metaphone, porém ambos são incapazes de efetuar uma busca eficaz em um banco de dados brasileiro devido ao número de acentos e caracteres especiais tais como cedilha. Por esse motivo o autor mostra como desenvolver uma rotina que se adapte a realidade dos bancos de dados brasileiros.

A idéia consiste em criar uma DLL que mais tarde será usada como UDF no banco de dados Firebird. Contudo o código da **Listagem 1** passado na edição 82 possui alguns erros detectados mais tarde pela nossa equipe técnica.

Uma das primeiras correções diz respeito ao uso do caractere '%' – o algoritmo da listagem 1 não considera este caractere como máscara na pesquisa (por exemplo, uma busca por 'adriano%santos' não retornaria nenhum resultado). Para adicionar esse recurso inclua o seguinte código ao *case..of* (linha 89):

```
'%': novo := novo + '%';
```

Outro erro detectado tem a ver com o código seguinte (linha 142):

```
novo := novo + ' ';
```

O caractere espaço adicionado a variável novo impossibilita a ação da função confundindo-a. Para resolver o problema basta apagar a linha 142 da função.

Em seguida o trecho de código que faz a retirada de caracteres especiais e acentos foi alterada para trabalhar também com caracteres minúsculos e acentuados (linhas 19 a 29). Veja na **Listagem 2** o novo trecho.

Listagem 1. A unit untMain

```
001. unit untMain;
002. interface
003. uses
004.   SysUtils;
005.
006. function CodiFonPT_BR(nome: PChar): PChar;
007.   cdecl; export;
008. implementation
009.
010. function CodiFonPT_BR(nome: PChar): PChar;
011. var
012.   i, p: integer;
013.   novo, aux: string;
014. begin
015.   try
016.     aux := AnsiUpperCase(nome);
017.     novo := '';
018.     { Tira acentos e cedilha }
019.     for i := 1 to Length(aux) do
020.     begin
021.       case aux[i] of
022.         'Á', 'Â', 'Ã', 'À', 'Ä', 'Å': aux[i] := 'A';
023.         'É', 'Ê', 'È', 'Ë': aux[i] := 'E';
024.         'Í', 'Î', 'Ì', 'Ï': aux[i] := 'I';
025.         'Ó', 'Ô', 'Õ', 'Ò', 'Ö': aux[i] := 'O';
026.         'Ú', 'Û', 'Ù', 'Ü': aux[i] := 'U';
027.         'Ç': aux[i] := 'C';
028.         'Ñ': aux[i] := 'N';
029.         'Ý', 'Ÿ', 'Y': aux[i] := 'I';
030.       else
031.         if Ord(aux[i]) > 127 then
032.           aux[i] := #32;
033.       end;
034.     end;
035.     { Retira E, DA, DAS, DE, DI, DO e DOS do nome
036.       José da Silva = José Silva
037.       João Costa e Silva = João Costa Silva }
038.     p := Pos(' DA ', aux);
039.     while p > 0 do
040.     begin
041.       Delete(aux, p, 3);
042.       p := Pos(' DA ', aux);
043.     end;
044.     p := Pos(' DAS ', aux);
045.     while p > 0 do
046.     begin
047.       Delete(aux, p, 4);
048.       p := Pos(' DAS ', aux);
049.     end;
050.     p := Pos(' DE ', aux);
051.     while p > 0 do
052.     begin
053.       Delete(aux, p, 3);
054.       p := Pos(' DE ', aux);
055.     end;
056.     p := Pos(' DI ', aux);
057.     while p > 0 do
058.     begin
059.       Delete(aux, p, 3);
060.       p := Pos(' DI ', aux);
061.     end;
062.     p := Pos(' DO ', aux);
063.     while p > 0 do
064.     begin
065.       Delete(aux, p, 3);
066.       p := Pos(' DO ', aux);
067.     end;
068.     p := Pos(' DOS ', aux);
069.     while p > 0 do
070.     begin
071.       Delete(aux, p, 4);
072.       p := Pos(' DOS ', aux);
073.     end;
074.     p := Pos(' E ', aux);
075.     while p > 0 do
076.     begin
077.       Delete(aux, p, 2);
078.       p := Pos(' E ', aux);
079.     end;
080.
081.     { Retira letras duplicadas
082.       Elizabette = Elizabete }
083.     for i := 1 to Length(aux)-1 do
084.       if aux[i] = aux[i+1] then
085.         Delete(aux, i, 1);
086.
087.     for i := 1 to Length(aux) do
088.   begin
089.     case aux[i] of
090.       { 'A','E','I','O','U','Y','H' e espaços
091.         ignora }
092.       'B','D','F','J','K','L','M','N','R','T',
093.       'V','X':
094.         novo := novo + aux[i];
095.       'C':      { CH = X }
096.         if aux[i+1] = 'H' then
097.           novo := novo + 'X'
098.         else   { Carol = Karol }
099.         if aux[i+1] in ['A','O','U'] then
100.           novo := novo + 'K'
101.         else   { Celina = Selina, Cintia = Sintia }
102.         if aux[i+1] in ['E','I'] then
103.           novo := novo + 'S';
104.       'G': { Jeferson = Geferson }
105.         if aux[i+i] = 'E' then
106.           novo := novo + 'J'
107.         else
108.           novo := novo + 'G';
109.
110.       'P': { Phelipe = Felipe }
111.         if aux[i+1] = 'H' then
112.           novo := novo + 'F'
113.         else
114.           novo := novo + 'P';
115.       'Q': { Keila = Queila }
116.         if aux[i+1] = 'U' then
117.           novo := novo + 'K'
118.         else
119.           novo := novo + 'Q';
120.       'S':
121.         case aux[i+1] of
122.           'H': { SH = X }
123.             novo := novo + 'X';
124.           'A','E','I','O','U':
125.             if aux[i-1] in ['A','E','I','O','U'] then
126.               { S entre duas vogais = Z }
127.               novo := novo + 'Z'
128.             else
129.               novo := novo + 'S';
130.         end;
131.       'W': { Walter = Valter }
132.         novo := novo + 'V';
133.       { no final do nome tem som de S ->
134.         Luiz = Luis }
135.       'Z':
136.         if (i = Length(aux)) or (aux[i+1] = ' ') then
137.           novo := novo + 'S'
138.         else
139.           novo := novo + 'Z';
140.     end;
141.   end;
142.   novo := novo + ' ';
143.   CodiFonPT_BR := PChar(novo);
144. except
145.   CodiFonPT_BR := PChar('');
146. end;
147. end;
148. end.
```

Nos trechos seguintes, estamos fazendo verificações para interpretar melhor a fonética de "*SS*" e "*Ç*" nas frases. Então acrescente um novo elemento ao *case..of* (linha 89) principal conforme a seguir:

```
'Ç': novo := novo + 'S';
```

E em seguida adicione antes do *case..of* o trecho da **Listagem 3**.

E por fim, outra grande mudança no código foi a forma como o sistema trata os primeiros caracteres da palavra. Por exemplo, no script anterior quando tínhamos o caso:

```
AMARAL DE SOUZA
MARIO LUIS
MARILENE  LOZADA
```

E efetuávamos a busca por 'AMARAL' o resultado era como a seguir:

```
AMARAL DE SOUZA = MRLSZ
MARIO LUIS = MRLS
MARILENE LOZADA = MRLNLZD
AMARAL = MRL
```

Observe que é estranho uma busca por 'AMARAL' retornar nomes como 'Mario' ou 'Marilene' – a pronúncia entre as palavras não é similar. Isso acontece porque o algoritmo original remove todas as vogais da palavra inteira; Assim, AMARAL é igual a MRL e MARILENE é igual a MRLN. A solução foi não remover a primeira vogal da palavra, deixando a pesquisa fonética mais precisa; Veja o exemplo a seguir:

```
AMARAL DE SOUZA = AMRLSZ
MARIO LUIS = MARLS
MARILENE LOZADA = MARLNLZD

AMARAL = AMRL
MARILENE = MARLN
MARIO = MAR
```

Nessa nova versão do algoritmo apenas o nome AMARAL seria retornado do conjunto de dados de exemplo. O código responsável por essa mudança pode ser visto a seguir:

```
else
  if ((i=1) or (i=2)) and (aux[i] in
    ['A','B','C','D','E','F','G','H','I','J',
     'K','L','M','N','O','P','Q','R','S','T',
     'U','V','W','X','Y','Z','0','1','2','3',
     '4','5','6','7','8','9'])
  then
    novo := novo + aux[i];
```

Na **Listagem 4** você pode conferir a listagem com todas as alterações.

Durante a confecção desta errata percebemos que o algoritmo pode ser evoluído de várias formas, deixando a pesquisa fonética ainda mais precisa. Se você fizer alguma melhoria nesse código fique a vontade para nos enviar - Publicaremos sua versão na revista ou no site DevMedia! ●

Listagem 2. Trecho modificado para trabalhar melhor com minúsculas acentuadas

```
for i := 1 to Length(aux) do
begin
  case aux[i] of
    'á','Á','Â','â','Ã','ã','À','à','Ä','ä': aux[i]:= 'A';
    'é','É','ê','Ê','è','È','ë','Ë': aux[i] := 'E';
    'í','Í','î','Î','ì','Ì','ï','Ï': aux[i] := 'I';
    'ó','Ó','ô','Ô','ò','Ò','õ','Õ','ö','Ö': aux[i] := 'O';
    'ú','Ú','û','Û','ù','Ù','ü','Ü': aux[i] := 'U';
    'ç','Ç': aux[i] := 'C';
    'ñ','Ñ': aux[i] := 'N';
    'ý','Ý','Y': aux[i] := 'I';
  else
    if Ord(aux[i]) > 127 then
      aux[i] := #32;
  end;
end;
```

Listagem 3. Trecho para tratamento de "SS"

```
for i:=1 to length(aux)-1 do
  if (aux[i] = aux[i+1]) then
  begin
    Delete(aux,i,1);
    if (aux[i]='S') and (aux[i+1] in ['I','E']) then
      aux[i]:='C';
    if (aux[i]='S') and (aux[i+1] in ['O','U']) then
    aux[i]:='Ç';
  end;
```

Listagem 4. Código completo com as devidas alterações

```
unit untMain;
interface
uses
  SysUtils;

function CodiFonPT_BR(nome: PChar): PChar;
var
  i, p: Integer;
  novo, aux: string;
begin
  try
    aux := AnsiUpperCase(nome);
    novo := '';
    for i := 1 to Length(aux) do
    begin
      case aux[i] of
        'á', 'Á', 'â', 'Â', 'ã', 'Ã', 'à', 'À', 'ä', 'Ä',
          'ª': aux[i] := 'A';
        'é', 'É', 'ê', 'Ê', 'è', 'È', 'ë', 'Ë':
          aux[i] := 'E';
        'í', 'Í', 'î', 'Î', 'ì', 'Ì', 'ï', 'Ï':
          aux[i] := 'I';
        'ó', 'Ó', 'ô', 'Ô', 'ò', 'Ò', 'ø', 'Õ', 'ö',
          'Ö': aux[i] := 'O';
        'ú', 'Ú', 'û', 'Û', 'ù', 'Ù', 'ü', 'Ü':
          aux[i] := 'U';
        'ç', 'Ç': aux[i] := 'C';
        'ñ', 'Ñ': aux[i] := 'N';
        'ý', 'Ý', 'Y': aux[i] := 'I';
      else
        if Ord(aux[i]) > 127 then
          aux[i] := #32;
      end;
    end;
    p := Pos(' DA ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 3);
      p := Pos(' DA ', aux);
    end;
    p := Pos(' DAS ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 4);
      p := Pos(' DAS ', aux);
    end;
    p := Pos(' DE ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 3);
      p := Pos(' DE ', aux);
    end;
    p := Pos(' DI ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 3);
      p := Pos(' DI ', aux);
    end;
    p := Pos(' DO ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 3);
      p := Pos(' DO ', aux);
    end;
    p := Pos(' DOS ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 4);
      p := Pos(' DOS ', aux);
    end;
    p := Pos(' E ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 2);
      p := Pos(' E ', aux);
    end;
    for i := 1 to length(aux) - 1 do
      if (aux[i] = aux[i + 1]) then
      begin
        Delete(aux, i, 1);
        if (aux[i] = 'S') and (aux[i + 1]
          in ['I', 'E']) then
          aux[i] := 'C';
        if (aux[i] = 'S') and (aux[i + 1]
          in ['O', 'U']) then
          aux[i] := 'Ç';
      end;
    for i := 1 to length(aux) do
    begin
      case aux[i] of
        'B', 'D', 'F', 'J', 'K', 'L', 'M', 'N', 'R',
        'T', 'V', 'X':
          novo := novo + aux[i];
        'C':
          if aux[i + 1] = 'H' then
            novo := novo + 'X'
          else if aux[i + 1] in ['A', 'O', 'U'] then
            novo := novo + 'K'
          else if aux[i + 1] in ['E', 'I'] then
            novo := novo + 'S';
        'G':
          if aux[i + 1] in ['E'] then
            novo := novo + 'J'
          else
            novo := novo + 'G';
        'P':
          if aux[i + 1] in ['H'] then
            novo := novo + 'F'
          else
            novo := novo + 'P';
        'Q':
          if aux[i + 1] in ['U'] then
            novo := novo + 'K'
          else
            novo := novo + 'Q';
        'S':
          case aux[i + 1] of
            'H': novo := novo + 'X';
            'A', 'E', 'I', 'O', 'U':
              if aux[i - 1] in
                ['A', 'E', 'I', 'O', 'U'] then
                novo := novo + 'Z'
              else
                novo := novo + 'S';
          else
            novo := novo + 'S';
          end;
        'Ç': novo := novo + 'S';
        'W': novo := novo + 'V';
        'Z':
          if (i = length(aux)) or (aux[i + 1] = ' ')
            then
            novo := novo + 'S'
          else
            novo := novo + 'Z';
        '%': novo := novo + '%';
      else
        if ((i = 1) or (i = 2)) and (aux[i]
          in ['A', 'B', 'C', 'D', 'E', 'F',
            'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
            'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
            'W', 'X', 'Y', 'Z', '0', '1', '2', '3',
            '4', '5', '6', '7', '8', '9']) then
          novo := novo + aux[i];
      end;
    end;
    CodiFonPT_BR := PChar(novo);
  except
    CodiFonPT_BR := PChar('');
  end;
end;
end.
```

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Desenvolvendo uma Aplicação Completa com PocketStudio

Aprenda a criar aplicações para o sistema operacional PalmOS – Parte 1

Ricardo C. Boaro
(rboaro@aquasoft.com.br)
trabalha com desenvolvimento de sistemas em Delphi há mais de 10 anos e PocketStudio há 3 anos. Atualmente é gerente de informática na Di Hellen Indústria de Cosméticos, e atua como instrutor certificado Borland na Aquasoft Tecnologia da Informação parceira da Borland, em Porto Alegre – RS. Borland Instrutor, Delphi 7, 2007 e Certified.

A partir deste artigo, criaremos uma aplicação para PalmOS completa utilizando a ferramenta PocketStudio. Algumas considerações são importantes em relação à programação para Palm Top. Atualmente existem *Palms* com sistema operacional Windows Mobile chamados Pocket PC's e os dispositivos com PalmOS. Existem muitas diferenças entre eles, mas a principal a ser entedida, é justamente a questão do Sistema Operacional.

Nesse mini-curso entenderemos as diferenças entre os sistemas operacionais PalmOS e Windows Mobile, conceitos de desenvolvimento móvel e exemplos práticos.

Criaremos uma aplicação voltada ao mercado de vendas, onde os representantes poderão cadastrar novos Clientes, Produtos e Pedidos. Em seguida poderão enviar os dados a matriz através da aplicação gerada e instalada no dispositivo móvel.

Conceitos, história e diferenças

PalmOS não é Windows, muitos imaginam que o PalmOS é uma versão do Windows para a plataforma móvel. Terrível engano, pois a estrutura interna é totalmente diferente e as chamadas às APIs também. Palms são aparelhos com menos memória que os Pocket PC's, o que não significa queda de performance pois a API do Palm compensa a diferença. O sistema operacional PalmOS ocupa em torno de 4 MB da memória, contra os mais de 200MB do Windows Mobile.

O custo dos portáteis equipados com PalmOS é bem menor que os com Windows Mobile. Certamente isso pesa na balança na hora em que empresas optam por informatizar sua equipe de vendas. Com o PocketStudio desenvolvemos aplicações para PalmOS, isso precisa estar bem claro. Um sistema desenvolvido com o PocketStudio não roda na API do Pocket PC.

PocketStudio não é Delphi, apesar de

algumas semelhanças com o IDE do Delphi 7 e anteriores, a forma de programar é diferente. Enquanto em sistemas desenvolvidos em Delphi trabalhamos com a linguagem Object Pascal, no PocketStudio usamos o Pascal puro. Após alguns esclarecimentos importantes vamos aos passos para instalação do PocketStudio e do PalmOS Emulator.

Instalando o emulador

Para testarmos nossas aplicações podemos instalar o PalmOS Emulator da empresa Access. O Emulador é provido de um arquivo executável, imagens de dispositivos e skins para tornar ainda mais atrativos os testes. Vamos efetuar o download do emulador pelo site Palm Brasil, que contém uma grande quantidade de programas e utilitários para Palm. O portal é parceiro da empresa Access mencionada anteriormente e mantém em sua área de downloads o PalmOS Emulator bem como imagens ("ROM's") dos aparelhos e skins.

Acesse o link *www.palmbrasil.com.br* e clique no link *Programs* na seção *downloads*. Logo em seguida abra o link *Palm Desktop e Simulador* na seção *Utilitários*. Na página 2 você encontrará o item *PalmOS Emulator 3.5* para download e mais abaixo as *ROM's* para download. Se preferir acesse diretamente o link *www.palmbrasil.com.br/programas/palmos2.html*.

Para efetuar o download é necessário ter cadastro e estar devidamente logado no site da Access. Para se cadastrar acesse o link *https://www.developerpavilion.com/palmos* e logo em seguida em *Register*. Preencha seus dados e faça o login. Logo após retorne ao site da Palm Brasil e clique nos links para download.

Você fará o download do arquivo *emulador-win.zip*.Também é interessante baixar o arquivo de *skins*, *PalmOS Emulator Skins 1.9* ("emulator-skins-19.zip"), presente nessa mesma página. Os *skins* são usados para visualizar o aparelho no emulador e no ambiente de desenvolvimento do PocketStudio como veremos mais adiante. Após ter feito o download de todos os arquivos necessários, descompacte-os na pasta criada e execute o arquivo *Emulador.exe*.

No caso dos *skins*, uma pasta *Skins_v1.9* será criada ao descompactar. É necessário renomear a pasta *Skins*. Nas subpastas desse diretório existem alguns arquivos de imagem em formato JPEG. Copie-os para a pasta *Skins*. Eles serão usados para mostrar o dispositivo no ambiente de desenvolvimento. Feito isso execute o arquivo *Emulador.exe*, ele lhe perguntará se deseja criar um atalho no menu *Programas*. Confirme e indique em qual pasta deseja que o emulador crie o atalho.

Vamos falar um pouco sobre o emulador que servirá como ambiente de testes em nosso *Desktop*. Vamos iniciá-lo e dar uma olhada nas configurações necessárias para o funcionamento correto de nossa aplicação. A primeira vez que o iniciarmos, uma tela de configuração é exibida (**Figura 1).**

O título da janela *New Session* indica que a partir dela criaremos uma nova seção para trabalharmos em paralelo a nossa aplicação. Seção nesse caso serão todas as configurações que fizermos na tela de abertura e tudo que gravarmos no emulador, como por exemplo a nossa aplicação. Nos exercícios que desenvolveremos nos próximos artigos ficará bem entendido o que é e como utilizar uma seção no emulador. Na **Figura 1** podemos ver quatro opções de configuração:

• *ROM file:* ROM é o modelo do dispositivo que iremos emular. Para cada modelo de Palm existe uma ROM para utilizarmos no emulador, A ROM que utilizaremos em nosso exercício esta gravada dentro do diretório de instalação do Emulador em uma pasta ROM's. Para a nossa aplicação selecione a ROM mostrada na figura. *Platform41_m505_enUS.rom;*

• *Device:* em *Device* irão aparecer todos os equipamentos Palm compatíveis com a ROM selecionada. Para o nosso exemplo escolha *Palm m505;*

• *Skin*: são as aparências disponíveis para o modelo que selecionamos em *Device*. Selecione a opção *Standard-English;*

• RAM size: você pode escolher a quantidade de memória que deseja utilizar na seção. Também será variável de acordo com o modelo do Palm escolhido, para o nosso exercício selecione *16,384K*;

Após todas as opções selecionadas clicamos em *OK*. E o emulador é carregado (**Figura 2**) exatamente igual ao modelo do Palm selecionado. Ele possui todas as funções do Palm verdadeiro, até a escrita *graffite* funciona como se estivéssemos escrevendo com a caneta.

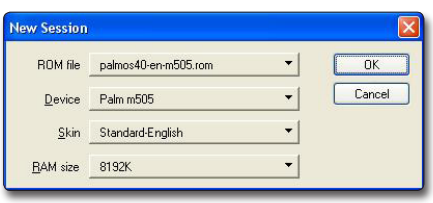


Figura 1. Primeira tela do Emulador

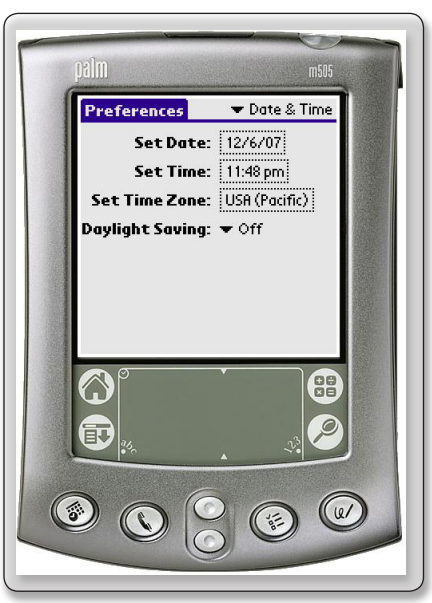


Figura 2. Emulador do Palm funcionando

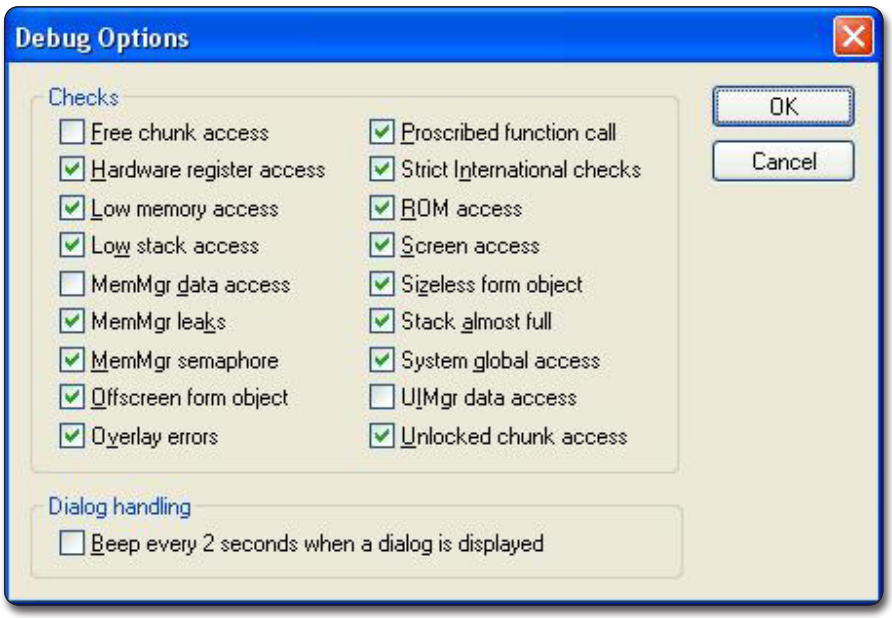


Figura 3. Configurações de Debug do Emulador

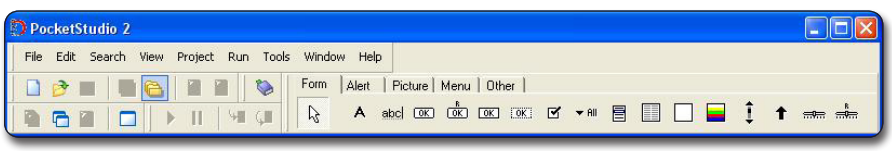


Figura 4. Menu principal e a paleta de componentes

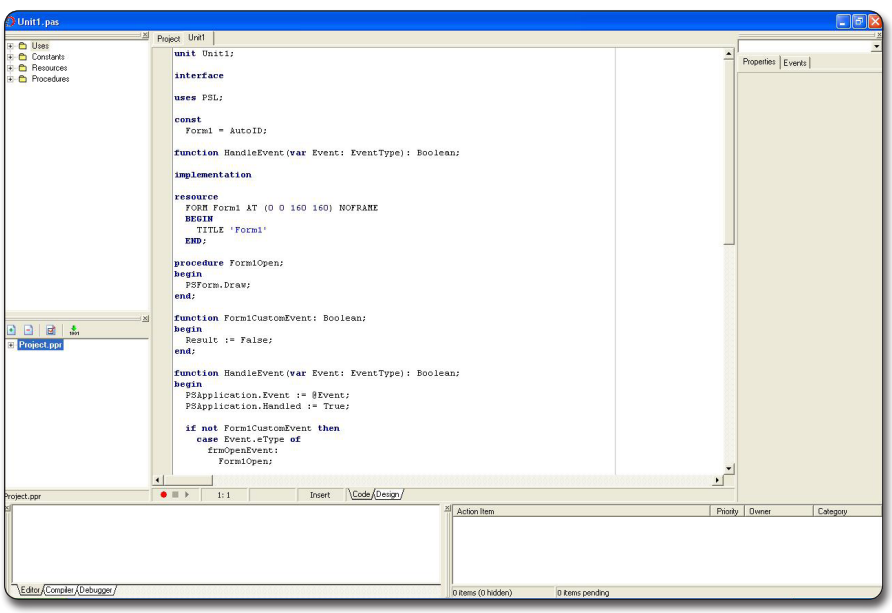


Figura 5. Janela principal do PocketStudio

Nota: A escrita Graffiti é um sistema de reconhecimento de escrita utilizado em todos os PDA's da plataforma PalmOS até a versão 4.1, baseado no alfabeto simples, onde em realidade não chega a ser um "reconhecimento da escrita" de verdade, pois o usuário precisa aprender as combinações de cada letra do alfabeto, símbolos, pontuações etc.

Antes de iniciarmos nossa aplicação é necessário que façamos mais algumas configurações em nosso emulador para evitarmos problemas. Clicando com o botão direito do mouse sobre ele navegue até a opção *Settings* e escolha o item *Debugging*. Aberta a caixa de diálogo desmarque os itens *Free chunk access*, *MemMgr data access* e *UIMgr data access* como mostrado na **Figura 3.**

Essas opções que desmarcamos, existem no emulador para futuras versões do sistema operacional do PalmOS, sendo assim, na versão atual precisam ser desmarcadas para evitar erros de *runtime* quando rodarmos o projeto no Pocket Studio. Muitos desenvolvedores perdem horas tentando encontrar o problema de sua aplicação, quando na verdade é apenas a configuração do emulador.

Instalando e conhecendo o PocketStudio

No site da WinSoft podemos baixar uma versão de avaliação do PocketStudio. Acesse o link *www.winsoft.sk/pstudio.htm*, encontre a seção *Products* e em seguida PocketStudio. No final da página encontre o link *PocketStudio 2.1 Standard 45-day trial version*. Será solicitado que entre com *Nome*, *Empresa* e *E-mail*. Os dados e instruções para *download* virão em seu endereço eletrônico, por isso informe um endereço válido.

A instalação é simples, basta executar o *Setup*.exe. Serão apresentadas algumas telas, confirmando o local da instalação, se será criado um atalho etc. Basta confirmar as mesmas, no final clicamos em *Finish* e o PocketStudio está instalado e pronto para ser utilizado. Recentemente instalei-o no Windows Vista e funcionou perfeitamente.

Vamos conhecer o IDE do PocketStudio

para depois iniciarmos o desenvolvimento de nossa aplicação. Nas **Figuras 4 e 5** podemos ver o menu principal juntamente com a paleta de componentes e ainda a janela principal da ferramenta, onde encontram-se as janelas *Code Editor*, *Project Manager*, *Object Inspector* entre outras.

As janelas que aparecem no ambiente são:

• No canto superior esquerdo está a janela do *Code Explorer*, onde podemos acessar as *procedures* da *Unit*;

• Logo abaixo está localizada a janela *Project Manager*;

Essa janela contém todas as *Units* do projeto;

• No canto inferior esquerdo está a janela *Messages* onde as mensagens e erros de complicação são mostradas;

• No canto inferior direito está a janela *To-Do-List* ("Lista de tarefas") que é um modo de controlarmos as implementações a serem feitas na aplicação;

• No centro temos o editor de código ("Code Editor") do PocketStudio, onde cada *Unit* é representado por uma aba. Pressionando *F12* alternamos entre o *Code Editor* e o formulário;

• Finalmente no canto superior direito está o *Object Inspector*, acessível pela tecla *F11* como no Delphi. Nessa janela configuramos as propriedades e eventos dos objetos visuais de nossa aplicação;

Todas essas janelas aparecem ancoradas por padrão, mas podem ser fechadas ou configuradas como você preferir.

No PocketStudio, uma configuração que considero útil é podermos mostrar o *skin* do Palm diretamente na IDE como mostrado na **Figura 6.** *Skin* é a imagem do dispositivo propriamente dita. Para isso clicamos em *Tools|Environment Options>Designer*. Marque a opção *Display skin* e em seguida escolha o *skin* que desejar. No campo *Path to skins* configure o caminho onde as imagens foram salvas. É importante fazer o download dessas imagens, caso contrário não será possível visualizá-las.

Ao voltamos ao PS temos o formulário já com a *skin* posicionada, isso facilita muito para montarmos o layout da tela da nossa aplicação. Voltando ao *Environment Options*, vamos dar uma olhada na aba *Debugger* que possui configurações importantes relacionadas ao emulador. Na opção *Emulator* informamos o local onde está o executável do emulador para que o PocketStudio possa rodar nossas aplicações. Se você instalou o PocketStudio e o PalmOS Emulator nos diretórios indicados na instalação, isso é configurado por *Default*. Caso tenha mudado o diretório, principalmente do Emulador é importante configurar esse item, caso contrário o PocketStudio irá encontrar o emulador.

Construindo a aplicação

Todas as configurações prontas, vamos iniciar o desenvolvimento de nossa aplicação. Nosso objetivo é criar uma aplicação que servirá de apoio à vendas, ou seja, os representantes de uma empresa irão utilizá-la para digitar e enviar os pedidos à empresa. Imaginemos uma empresa que possui vários representantes e os mesmos fazem pedidos diariamente. Estes são enviados para a empresa por fax. Além de ter um custo elevado, a empresa ainda tem o trabalho de digitar todos os pedidos em seu ERP.

Nossa aplicação, além de diminuir consideravelmente o custo de envio dos pedidos, irá agilizar o processo, pois não será mais necessário digitar os pedidos, os mesmo serão importados do nosso dispositivo diretamente para o ERP.

Entendido o objetivo de nosso exercício iremos começar pela tela principal, onde criaremos o menu e alguns botões para acessarmos os cadastros e telas de consulta da nossa aplicação.

Abra o PocketStudio e crie uma nova aplicação em *File|New>Application*. O PocketStudio abrirá uma nova aplicação e será apresentado um formulário em branco. Antes de começarmos a montar o layout vamos salvar nossa aplicação. Acesse o menu *File|Save as* e salve o formulário como "uPrincipal.pas" e o projeto como "Vendas.ppr" em uma pasta de sua preferência. Feito isso vamos alterar algumas propriedades do formulário principal.

No *Object Inspector* modifique a propriedade *Title* para "Vendas" e seu *Name* para "FrmPrincipal". Após isso pressione *Ctrl + F9* para compilar a nossa aplicação e note que erros foram apresentados, ao todo dois. Esse erros

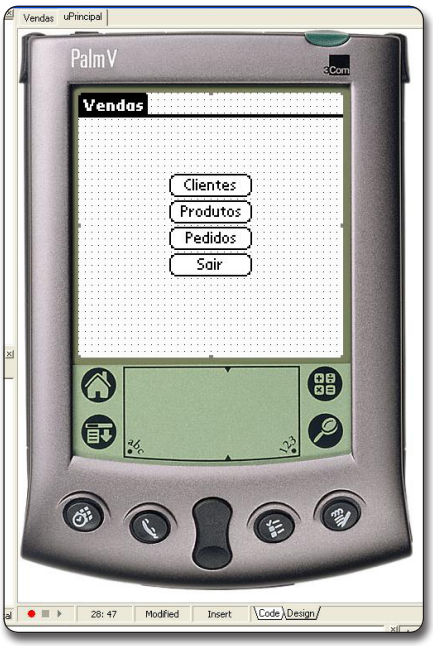


Figura 6. Imagem do dispositivo no IDE

foram causados pela alteração do nome de nosso formulário. Apenas mude de *Form1* para "FrmPrincipal" nos dois locais onde o compilador apontar.

Antes de continuarmos com o desenvolvimento de nossa aplicação vamos dar uma olhada no diretório onde foi salvo nosso projeto para entendermos os tipos de arquivos criados pelo PocketStudio.

Note que mesmo após compilarmos nossa aplicação não temos um arquivo com a extensão .exe comum em aplicações desktop. Isso por que no PalmOS os executáveis têm extensão .prc.

Voltando ao PocketStudio, o primeiro passo da nossa aplicação é criarmos os botões de acesso aos cadastros e telas de consulta e também o botão para fecharmos a aplicação. Nossa aplicação terá os formulários de *Clientes*, *Produtos*, *Pedidos* e *Itens de Pedidos*.

Inclua quatro *Buttons* da paleta *Form*. Os nomes para cada botão são: *btnClientes*, *btnProdutos*, *btnPedidos* e *btnSair*, respectivamente. Aumente a largura do primeiro botão digitando 50 na propriedade *Width*. Agora vamos aumentar a largura dos demais botões de uma só vez. Para isso selecione todos os botões pressionando a tecla *Shift* e clicando em cada botão. Feito isso dê uma clique de direita na mouse em cima de um dos cinco botões

e escolha a opção *Size*. Marque o item *Grow to largest* no grupo de opções *Width* e confirme a caixa de diálogo. Com isso deixamos todos os botões com o tamanho do maior componente da seleção.

Com o grupo de botões ainda selecionados, clique novamente com o direito e escolha *Align*. Centralize-os no formulário marcando o último item de cada grupo de opções, ou seja, *Center in window*. Veja o layout sugerido na **Figura 7**.

Analisando a linguagem

Vamos começar a entender um pouco mais a maneira como o compilador do PocketStudio trabalha. Clicando na aba *Code* ou pressionando *F12*, vamos ao fonte de nossa *Unit* principal.

No início da *Unit* temos o nome de nosso formulário principal. Logo abaixo, na seção *Interface*, ficam definidos os métodos visíveis para as outras *Units* bem como a cláusula *Uses*, onde encontramos declarada a *Unit PSL*, que é a *PSLibrary*. Ela é um conjunto de bibliotecas em forma de arquivos *.pas* que simplificam o trabalho do desenvolvedor, pois tem várias funções para manipularmos objetos visuais e banco de dados no Palm. Podemos compará-la a VCL do Delphi. Mais adiante estudaremos essa biblioteca mais detalhadamente.

Em seguida nos deparamos com uma a constante *FrmPrincipal* do tipo *AutoID*. É o identificador para nosso formulário; Essa seção de constantes contém todos os identificadores para todos os objetos existentes em nossa *Unit*, visuais ou não. Note que o primeiro objeto a ser identificado é nosso formulário *FrmPrincipal*.

Em seguida temos uma função chamada de *HandleEvent*, que recebe como parâmetro um evento do tipo *EventType* e retorna um valor booleano. Todo formulário tem o seu próprio *EventHandle*. Essa função é responsável por controlar todos os eventos que chegam ao formulário. Após codificarmos nossos botões, voltaremos a ela para entendermos melhor o seu funcionamento (**Listagem 1).**

Mais adiante existe a seção *Implementation*, também encontrada em Delphi, onde são definidos todos os métodos e os objetos acessíveis apenas para essa *Unit*. Note que para cada botão que colocamos o PocketStudio gerou uma constante com o valor *AutoID* atribuído, ou seja, um identificador.

```
const
  btnClientes = AutoID;
  btnProdutos = AutoID;
  btnPedidos = AutoID;
  btnSair = AutoID;
```

Em *resource* temos os valores das propriedades configuradas no *Object Inspector*. Após codificarmos os eventos comuns para todos os formulários, falaremos mais sobre eles na seqüência de nosso exemplo.

Codificando o formulário

O primeiro botão que vamos codificar será o *btnSair*, para podermos finalizar nossa aplicação sem precisarmos clicar no ícone da *Home* representado por uma "casinha" no dispositivo. Clique duas vezes no *btnSair* e note que somos levados ao editor de código do Palm no evento *BtnSairSelect*. Codifique de acordo com o código da **Listagem 2**.

No código da **Listagem 2** declaramos uma variável do tipo *EventType*. Em seguida atribuímos o valor *appStopEvent* a sua propriedade *eType*. Por fim invocamos o método *EvtAddEventToQueue* passando

Listagem 1. Trecho da Unit uPrincipal

```
unit uPrincipal;
interface
uses PSL;
const
  FrmPrincipal = AutoID;
function HandleEvent(var Event: EventType): Boolean;
implementation
const
  btnClientes = AutoID;
  btnProdutos = AutoID;
  btnPedidos = AutoID;
  btnSair = AutoID;
resource
  FORM FrmPrincipal AT (0 0 160 160) NOFRAME
  BEGIN
    TITLE 'Vendas'
    BUTTON btnClientes 'Clientes' AT (56 50 48 12)
    BUTTON btnProdutos 'Produtos' AT (56 66 48 12)
    BUTTON btnPedidos 'Pedidos' AT (56 82 48 12)
    BUTTON btnSair 'Sair' AT (56 98 48 12)
  END;
```

Listagem 2. Código do botão Sair

```
procedure BtnSairSelect;
var
 Evt: EventType;
begin
 Evt.eType:= appStopEvent;
 EvtAddEventToQueue(@Evt);
end;
```

Listagem 3. Loop principal do projeto gerado pelo PocketStudio

```
procedure EventLoop;
var
  Event: EventType;
  Error: UInt16;
begin
  repeat
    EvtGetEvent(Event, evtWaitForever);
    if not SysHandleEvent(Event) then
      if not MenuHandleEvent(nil, Event, Error) then
        if not ApplicationHandleEvent(Event) then
          FrmDispatchEvent(Event); // sairá do loop quando receber um appStopEvent.

  until Event.eType = appStopEvent;
end;
```

Figura 7. Layout sugerido para a tela principal

para ele nossa variável. Isso terá efeito dentro da *Unit* principal do projeto onde encontramos o código da **Listagem 3.**

A variável *Evt*, declarada com o tipo *EventType* irá receber a atribuição *appStop Event* que indica ao sistema operacional do PalmOS que deve encerrar o *loop* que mantém a aplicação aberta. Em outras palavras, a aplicação será finalizada.

A **Listagem 3** possui o código completo do método *EventLoop* localizado no fonte principal do projeto, no nosso caso chamado de *Vendas*.

> **Nota**: A *Unit Vendas* se assemelha-se ao *Source* principal de um sistema desenvolvido em Delphi acessado pela opção *Project>View Source*.

Com ele fica fácil entendermos que ele estará em um loop, até que o atributo passado para ele seja do tipo *appStopEvent*, como mostrado na última linha do código. Compilamos nosso projeto com *Ctrl + F9*, se tudo estiver certo, salvamos o projeto clicando no menu *File>Save*. Para rodarmos a aplicação basta clicarmos em *F9*, e o PocketStudio automaticamente abrirá o emulador. Veja nossa aplicação funcionando na **Figura 8.**

Clicando em sair nossa aplicação é encerrada e o PocketStudio fica piscando na barra de menu indicando que recebeu uma chamada, no caso do nosso Emulador no momento em que finalizamos a aplicação.

Criando menus no PocketStudio

Continuando nosso exemplo, vamos criar um menu em nosso formulário principal. Os menus estão localizados na paleta *Menu* na barra de componentes do PS. O menu deve ser adicionado diretamente no *Code Editor* acima da seção *resource*. Então clique acima da palavra reservada *resource*, selecione o componente *Menu* na paleta e clique uma vez no local informado anteriormente. Você notará a inclusão de uma nova constante como mostrado a seguir:

```
const
  Menu1 = AutoID;
resource
  MENU Menu1
  BEGIN
  END;
```

Como podemos ver o PocketStudio declarou uma constante *Menu1 = AutoID* para o componente e uma área de *resource* com o nome do componente ("Menu1") e *begin..end*. Continuando a construir nosso menu abrimos uma linha entre o *Begin* e o *End* e adicionamos o segundo componente da paleta *Menu* que é o *PullDown*. Feito isso abrimos mais uma linha entre o *Begin* e *End* do *PullDown* e colocamos o próximo componente, um *Item*. Na **Listagem 4** veja código completo para após a criação do menu.

O próximo componente a ser adicionado chamado *Separator*, que é apenas um separador entre os menus como o próprio nome diz. Como teremos apenas um item de menu não será necessário adicioná-lo ao código. Feito isso estamos com o esqueleto de nosso menu criado. Após inserirmos os menus no código, vamos renomeá-los para entedermos bem o funcionamento de cada componente adicionado.

O primeiro adicionado foi o menu principal, e por *default* nomeado como *Menu1*, vamos renomeá-lo para *MnuAplicacao*.

Muita atenção na hora de renomear os componentes pois no PocketSudio precisamos renomear o *resource* do componente e na declaração da *const* do mesmo.

O outro componente adicionado foi o *PullDown*, vamos mudar seu nome de *PullDown* para *Arquivo*. Esse componente não tem *resourceID* pois é apenas o título do nosso menu e não terá nenhum efeito associado a ele.

Para finalizarmos nosso primeiro exemplo de menu falta renomearmos o componente *MenuItem1* para *MnuSair* e mudar seu texto associado de *MenuItem* para *Sair*. A letra *"T"* que está comentada entre chaves no código acima, funciona como uma tecla de atalho no Palm. Descomente-a para usarmos como tecla de atalho.

Para cada *Item* adicionado é necessário nomear e inserir uma referência a ele na área de constantes do menu. Por isso modifique de *MenuItem1* para *MnuSair* e crie uma constante como podemos ver a seguir na **Listagem 5**.

Como mencionado, declare uma constante como *AutoID* para o componente *MnuSair*. Isso é necessário pois o PS não declara automaticamente os itens de menu.

Feito isso, precisamos associar o menu ao formulário. Para isso pressione *F12* para que possamos ver o formulário. Em seguida *F11* ou vá direto ao *Object Inspector*. Na propriedade *Menu* selecione o *FrmPrincipal*.

Por fim falta dizer para o PS o que ele deve fazer com esse menu, ou melhor, qual será o efeito ao clicarmos com a caneta no menu *Sair*. Logicamente finalizaremos a aplicação, ou seja, faremos a codificação do evento responsável por executar as ações do *Menu*. Para isso clique duas vezes no evento *OnMenu* do formulário e digite o código da **Listagem 6**.

Observe que fizemos um *case..of* no *Item*, parâmetro recebido no evento, e verificamos qual menu foi clicado. Caso seja *MnuSair*, único item criado, executamos o procedimento de encerramento da aplicação.

Listagem 4. Criação do menu

```
const
  Menu1 = AutoID;
resource
  MENU Menu1 >>> Componente Menu
  BEGIN
    PULLDOWN 'PullDown' >>> Componente PullDown
    BEGIN
      MENUITEM MenuItem1 'MenuItem' {'T'} >>> Componente MenuItem
    END
  END;
```

Listagem 5. Menu criado e alterado

```
const
  MnuPrincipal = AutoID;
  { MnuSair será associado ao método menu do Formulário }
  MnuSair = AutoID;

resource
  { Componente Menu renomeado para MnuPrincipal }
  MENU MnuPrincipal
  BEGIN
    { Componente PullDown renomeado para Arquivo }
    PULLDOWN 'Arquivo'
    BEGIN
      { Componente MenuItem renomeado para Sair }
      MENUITEM MnuSair 'Sair' 'S'
    END
  END;
```

Listagem 6. Código do evento OnMenu

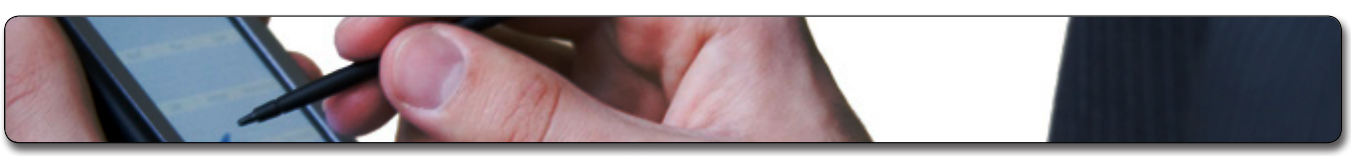
```
procedure FrmPrincipalMenu(Item: UInt16);
var
  Evt: EventType;
begin
  case Item of
    MnuSair:
      begin
        Evt.eType := appStopEvent;
        EvtAddEventToQueue(@Evt);
      end;
  end;
end;
```

Conclusão

Neste artigo tivemos o primeiro contato com o PocketStudio e o PalmOS Emulator, iniciamos nossa aplicação de vendas, e criamos a tela principal da aplicação. E também trabalhamos com botões e menus.

No próximo artigo iremos incrementar a tela principal, adicionando algumas consistências. Também veremos a tela de consulta de clientes, o que irá nos levar ao primeiro contato com *Banco de Dados* no PocketStudio, conceitos e formas de manipulação. Até o próximo artigo, e bons códigos a todos! ●

Figura 8. Sistema vendas funcionando no emulador



POR QUE A ALOG
É LÍDER EM HOSTING GERENCIADO?

- Dois Data Centers próprios – RJ e SP
- Projetos de Hosting customizados
- Foco em serviços de Data Center
- Gerência total dos servidores
- 850 clientes corporativos
- 8 mil m² de área construída
- Destaque do Ano no segmento Internet Serviços no Anuário Telecom 2007

EMPRESA CERTIFICADA ISO 9001 ANAB ACCREDITED BSI

EQUIPE CERTIFICADA EM:

PMI	PMP (PROJECT MANAGEMENT PROFESSIONAL)
ITIL	ITIL FOUNDATION
MICROSOFT	MCP (MICROSOFT CERTIFIED PROFESSIONAL), MCDST (MICROSOFT CERTIFIED DESKTOP SUPPORT TECHNICIAN), MCSE (MICROSOFT CERTIFIED SYSTEM ENGINEER)
CISCO	CCNA (CISCO CERTIFIED NETWORK ASSOCIATE) CCDA (CISCO CERTIFIED DESIGN ASSOCIATE)
RED HAT	RHCE (RED HAT CERTIFIED ENGINEER)
LPI INSTITUTE	LPI-1 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 1) LPI-2 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 2)
GIAC	GSEC (GIAC SECURITY ESSENTIALS GOLD)
CHECKPOINT	CCSA (CHECKPOINT CERTIFIED SECURITY ADMINISTRATOR)

ALOG
DATA CENTERS DO BRASIL

www.alog.com.br | 0800 282 3330

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

VCL for the Web e Ajax

Utilizando o poder do Ajax em aplicações com VCL for the Web no Delphi 2007

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Estamos entrando na era da *Web 2.0* e certamente uma das maiores inovações da nova internet, dentre muitas, tem um nominho bem pequeno, mas que supera muitas espectativas: Ajax. Atualizar páginas sem a necessidade de *reload* e requisições ao servidor tornou-se rotina em diversos sites. São inúmeras as possibilidades. Hoje podemos encontrar Ajax em portais como Yahoo!, UOL, Terra, Orkut entre outros.

No artigo "Delphi 2007 em Detalhes", na edição 86, publiquei diversos exemplos sobre os principais aspectos da nova IDE e entre elas estava a *VCL for the Web*. A *VCL for the Web* é o novo nome dado aos antigos componentes Intraweb da empresa Atozed Software.

Agora totalmente reformulada e com novos componentes, a *VCL for the Web* vem totalmente compatível com Ajax. Veremos nesse artigo onde, como e porque aplicar Ajax. Faremos a construção de uma página de cadastro onde alguns itens utilizarão Ajax e outros farão requisições ao servidor, assim poderemos visualizar as diferenças entre ambos métodos.

Entendendo o que é Ajax

Ajax, na verdade, não é uma tecnologia e sim um conjunto de tecnologias já conhecidas que unidas formam um novo conceito de desenvolvimento dinâmico e assíncrono. O objetivo principal do Ajax é tornar a navegação Web mais dinâmica e interativa. Esse novo modelo de desenvolvimento envolve algumas das principais tecnologias encontradas para o ambiente Web, tais como: XHTML e CSS, DOM, XML e XSLT, XMLHttpRequest e Javascript. Seu significado é *Asynchronous Javascript And XML*.

No modelo atual de internet, uma página é carregada após obter seus dados de uma requisição ao servidor *Web*, ou seja, toda e qualquer página só é atualizada mediante a um *reaload*, recarregamento da mesma. Um exemplo disso seria a

exclusão de um e-mail no *webmail*. Você seleciona a mensagem, clica em excluir, confirma e então a página é recarregada não mostrando mais o item, já que o mesmo foi apagado.

Em Ajax a etapa de recarregamento não acontece. Experimente selecionar uma mensagem no Gmail e apagá-la. Com certeza perceberá que somente o item sumiu sem que a página inteira fosse recarregada. Isso é Ajax. No diagrama da **Figura 1** podemos ver claramente o fluxo de trabalho que envolve uma ou mais páginas *Web*.

A interação com o usuário também é parte das inovações da *Web 2.0*. Com Ajax é possível interagir mais eficientemente com as ações do usuário/internauta. Um exemplo fortíssimo disso está no novo *webmail* do Yahoo! que agora implementa abas, menus de contexto, barras de progresso etc., como podemos ver na **Figura 2**.

De fato o Ajax chegou em boa hora, juntamente com o aumento da banda larga de modo geral. Vejamos então como implementar uma tela de cadastro fazendo uso do modelo utilizando a *VCL for the Web* no Delphi 2007 for Win32.

Criando o banco de dados

Criaremos uma base de dados em Firebird, bem simples contendo apenas duas tabelas: *Cursos* e *Concursos*. Para isso acesse o seu gerenciador preferido de banco de dados e crie um banco de dados chamado *Banco.fdb* com as tabelas citadas. A estrutura das tabelas deverá seguir a proposta das **Tabelas 1** e **2**.

Não será necessário fazer a criação de nenhuma chave primária ou *constraint*, pois deveremos nos concentrar apenas na aplicação exemplo e no Ajax. A idéia é que possamos cadastrar alguns cursos e que estes cursos dêem direito a prestar concurso para determinadas áreas, ou seja, um aluno formado em *Publicidade e Propaganda* poderá prestar concurso para *Designer Gráfico, Webdesigner, Publicitário* etc. Sendo assim cadastre alguns cursos e concursos conforme proposto na **Figura 3**.

Observe que o campo CONCURSO está ligado ao campo *ID* da tabela de concursos, ou seja, quem é formado em *Sistemas de Informação* ou *Tecnologia da Informação* poderá prestar concurso para *Analista de Sistemas* ou *Técnico em Informática*. Quem é formado em Direito poderá prestar para *Juiz criminal* ou *Auditor fiscal*.

É claro que poderíamos ter criado chaves, *primary keys*, *constraints*, *foreign keys* etc., mas para ilustrar este exemplo o que foi proposto é suficiente.

Se preferir execute o *script* da **Listagem 1** que contém a criação e inclusão de todos os registros nas respectivas tabelas.

Construindo um cadastro com VCL for the Web

Inicie uma nova aplicação *VCL for the Web* no Delphi 2007 for Win32 usando o menu *File|New>Other>VCL for the Web>VCL for the Web Application Wizard*. Na tela que se abre selecione a opção *StandAlone Application*. Em seguida dê o nome de "Cadastro" em *Project Name* e em *Project Directory* indique o caminho dos arquivos para gravação (**Figura 4**). Aguarde alguns instantes até que o Delphi crie os arquivos necessários para o desenvolvimento do projeto.

Perceba que o Delphi nos cria as *Units Cadastro, UserSessionUnit* e *Unit1*. O que nos interessa no momento é a *Unit1*. Clique nela e salve-a como "uPrincipal.pas" e seu *Name* modifique para "frmCadastro".

Em seguida desenhe uma tela como mostrado na **Figura 5** tendo em vista que

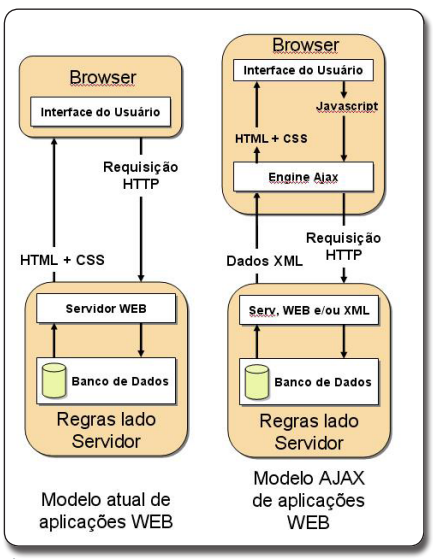


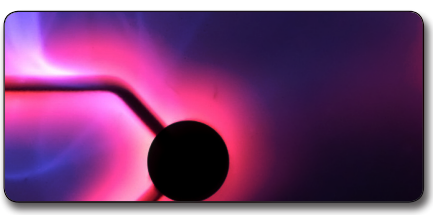
Figura 1. Fluxo de trabalho em uma página Web

Campo	Tipo/Tamanho
CURSO	VarChar(50)
CONCURSO	Integer

Tabela 1. Tabela de Cursos do sistema

Campo	Tipo/Tamanho
ID	Integer
CONCURSO	VarChar(50)

Tabela 2. Tabela de Concursos do sistema



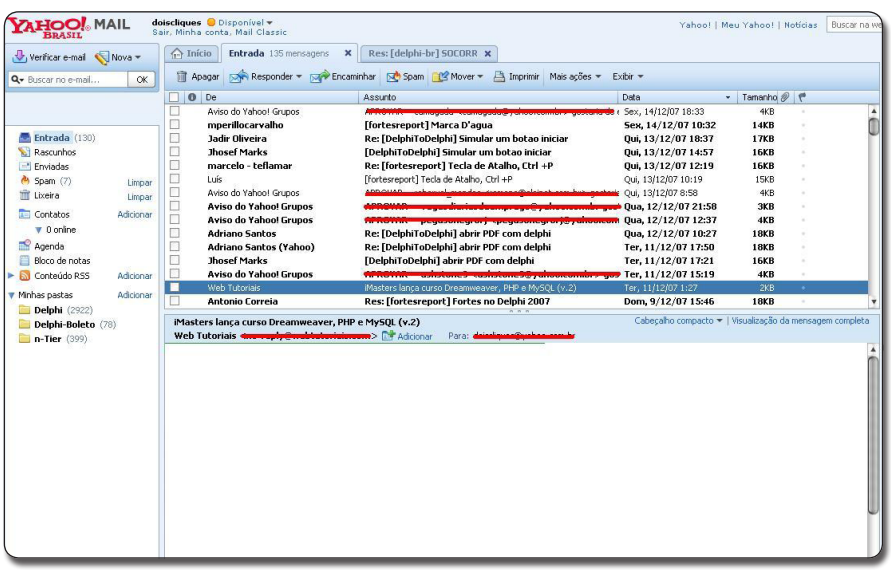


Figura 2. Site do Yahoo! Totalmente Ajax

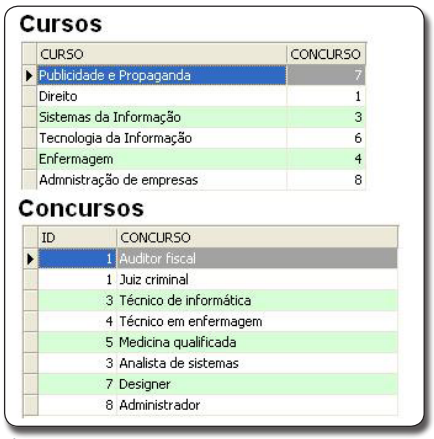


Figura 3. Proposta de registros a serem cadastrados

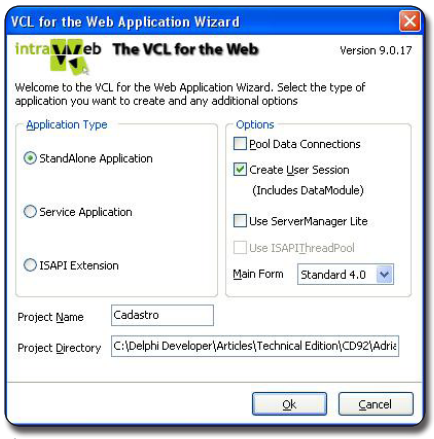


Figura 4. Criação do projeto VCL for the Web

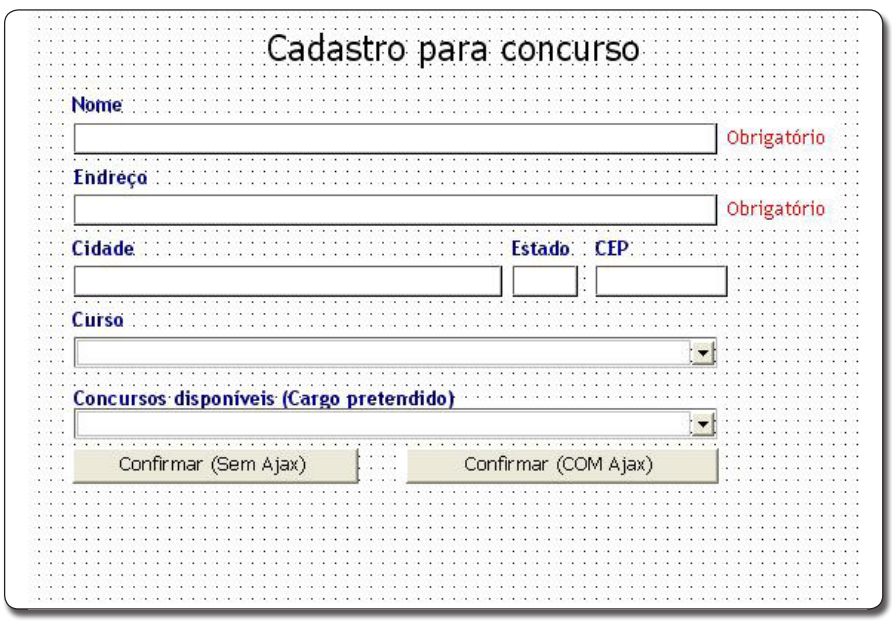


Figura 5. Exemplo de layout da tela de cadastro

Listagem 1. Criação completa do banco de exemplo

```
SET SQL DIALECT 3;
SET NAMES WIN1252;
CREATE DATABASE '<Caminho>\Banco.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
CREATE TABLE CONCURSOS (
    ID        INTEGER,
    CONCURSO  VARCHAR(70)
);
CREATE TABLE CURSOS (
    CURSO     VARCHAR(50),
    CONCURSO  INTEGER
);
INSERT INTO CONCURSOS (ID, CONCURSO) VALUES (1, 'Auditor fiscal');
INSERT INTO CONCURSOS (ID, CONCURSO) VALUES (1, 'Juiz criminal');
INSERT INTO CONCURSOS (ID, CONCURSO) VALUES (3, 'Técnico de informática');
INSERT INTO CONCURSOS (ID, CONCURSO) VALUES (4, 'Técnico em enfermagem');
INSERT INTO CONCURSOS (ID, CONCURSO) VALUES (5, 'Medicina qualificada');
INSERT INTO CONCURSOS (ID, CONCURSO) VALUES (3, 'Analista de sistemas');
INSERT INTO CONCURSOS (ID, CONCURSO) VALUES (7, 'Designer');
INSERT INTO CONCURSOS (ID, CONCURSO) VALUES (8, 'Administrador');
COMMIT WORK;
INSERT INTO CURSOS (CURSO, CONCURSO) VALUES ('Publicidade e Propaganda', 7);
INSERT INTO CURSOS (CURSO, CONCURSO) VALUES ('Direito', 1);
INSERT INTO CURSOS (CURSO, CONCURSO) VALUES ('Sistemas da Informação', 3);
INSERT INTO CURSOS (CURSO, CONCURSO) VALUES ('Tecnologia da Informação', 6);
INSERT INTO CURSOS (CURSO, CONCURSO) VALUES ('Enfermagem', 4);
INSERT INTO CURSOS (CURSO, CONCURSO) VALUES ('Admnistração de empresas', 8);
COMMIT WORK;
```

faremos uso apenas dos componentes da paleta *IWStandard*. Basicamente utilizei apenas *IWLabel's, IWEdit's, IWCombo Boxes* e *IWButton's*. Os nomes dos *IWLabel's* em vermelho são "lblNome" e "lblEndereco", respectivamente. Eles servirão de guia para que o usuário saiba quais campos são obrigatórios.

Modifique a propriedade *Font>Font Family* dos componentes para *Verdana, Helvetica, Sans-Serif*. Em seguida modifique sua propriedade *Color* para *clWebRed*.

> **Nota**: Repare que os nomes de grande parte das cores no *VCL for the Web* contém o prefixo *clWeb*. É importante lembrar que na internet há uma quantidade limitada de cores para se escolher, por isso opte, de preferência, utilizar-se das cores com o prefixo *clWeb*, pois é garantia de que a cor escolhida será corretamente visualizada no *Browser*.

Os *IWEdit's* renomeie-os para: "edtNome", "edtEndereco", "edtCidade", "edtEstado" e "edtCEP". Os "IWCombo Boxes" ficarão como "iwcbxCursos" e "iwcbxConcursos", respectivamente. Por fim os botões ganharão os nomes de "btnSemAjax" e "btnComAjax". O layout deverá se parecer com a **Figura 5**.

Como podemos perceber incluímos dois *ComboBoxes*, um com o nome de *iwcbxCursos* e o outro de *iwcbxConcursos*. Obviamente faremos uma conexão com o banco de dados e traremos todos os cursos disponíveis no primeiro *ComboBox*. À medida que o usuário final seleciona um dos *cursos*, o segundo *ComboBox* é carregado com os *concursos* disponíveis para o curso selecionado.

Vamos criar uma conexão com o banco de dados Firebird usando *dbExpress*. Insira uma componente *SQLConnection* ("sqlConexao") da paleta *dbExpress* e clique duas vezes nele. Na tela que se abre clique no botão "+" para criarmos uma nova conexão. Selecione o *driver* Interbase e dê um novo a conexão.

Após a confirmação preencha os atributos de conexão com o banco tais como: *DataBase, SQLDialect e ServerCharSet*, além das credenciais de acesso ao banco, por *default* usuário *SYSDBA* e senha *masterkey*.

Insira agora um componente *SQLDataSet*("sdsCursos") ligando-o ao *sqlConexao* através da propriedade *SQLConnection*. Sua propriedade *CommandText* deverá conter a instrução *SQL* como segue:

```
select * from Cursos
```

Repita esse passo incluindo outro *SQLDataSet*("sdsConcursos"). Insira o código *SQL* a seguir em sua propriedade *CommandText* como feito anteriormente.

```
select * from Concursos where ID =:ID
```

Agora insira dois componentes *DataSetProvider*, "dspCursos" e "dspConcursos", ambos apontados para seu respectivo *SQLDataSet* pela propriedade *DataSet*. Por último inclua dois componentes *ClientDataSet* com os nomes "cdsCursos" e "cdsConcursos". Ligue-os ao *dspCursos* e *dspConcursos*, respectivamente. Se preferir observe a **Figura 6** que possui os componentes de acesso inseridos.

Agora que configuramos a conexão com o banco de dados, faremos a codificação para trazermos os dados e atualizarmos os *ComboBoxes*. Clique duas vezes em uma área vazia do formulário e digite o código da **Listagem 2** no evento *OnCreate* do formulário principal.

Com certeza irá notar que não há nada de especial no código da **Listagem 2**. Apenas conectamos ao banco de dados, abrirmos a *query Cursos* e efetuamos um laço *while* incluindo todos os registros da tabela *Cursos*. A única coisa diferente que estamos fazendo é incluir o campo *ID* do *CONCURSO*, que é inteiro, no início da *string*. Ele será usado posteriormente para selecionar o(s) concurso(s) relacionado ao curso selecionado.

Codificando com Ajax

Agora vem a parte mais interessante do exemplo: vamos entender onde codificar nosso Ajax. Como explicado no artigo "Delphi em Detalhes", já citado no início deste artigo, não há segredos na utilização de Ajax, pois o Delphi 2007 for Win32 é totalmente compatível com ele.

Grande parte dos componentes da *VCL for the Web* possui eventos com o prefixo *OnAsync*. Esses eventos são controlados por meio de Ajax e nos possibilitam escrever código sem a necessidade de

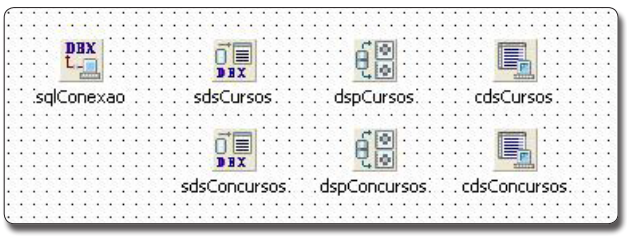


Figura 6. Componentes de acesso ao banco

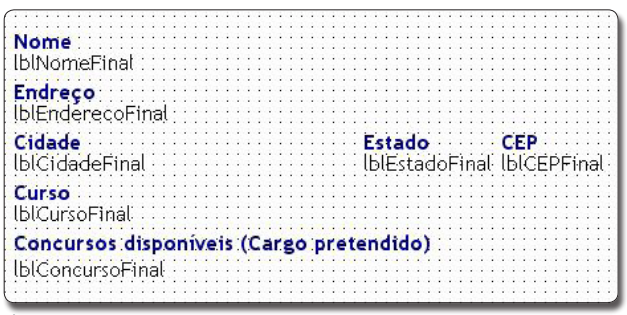


Figura 7. Layout de componentes para receberem valores

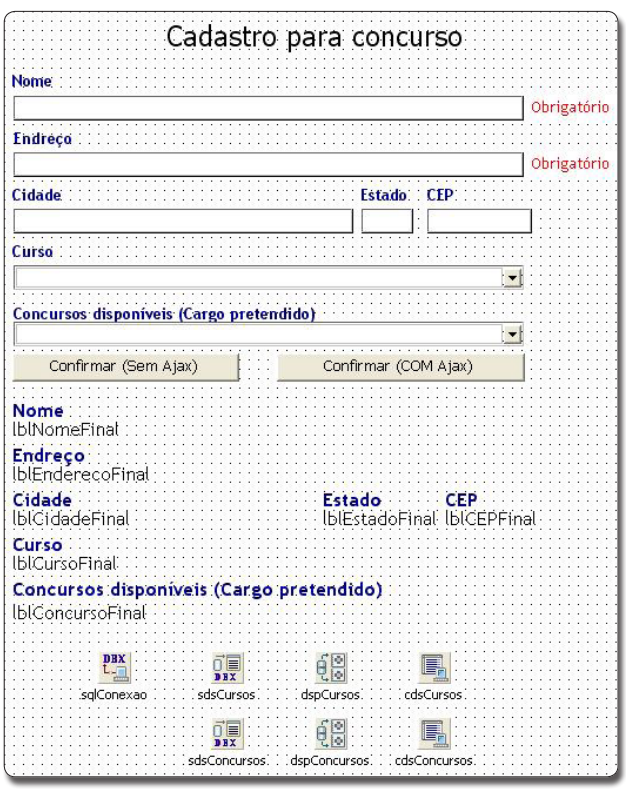


Figura 8. Layout final da página

Listagem 2. Código do evento OnCreate do formulário

```
procedure TfrmCadastro.IWAppFormCreate(Sender: TObject);
begin
  if not sqlConexao.Connected then
    sqlConexao.Connected := True;

  cdsCursos.Open;
  if not (cdsCursos.IsEmpty) then
  begin
    iwcbxCursos.Items.Clear;
    while not cdsCursos.EOF do
    begin
      iwcbxCursos.Items.Add(
        cdsCursos.FieldByName('CONCURSO').AsString + ' - ' +
        cdsCursos.FieldByName('CURSO').AsString);
      cdsCursos.Next;
    end;
  end;
  cdsCursos.Close;
end;
```

Listagem 3. Evento OnAsyncExit do edtNome

```
procedure TfrmCadastro.edtNomeAsyncExit(Sender: TObject; EventParams: TStringList);
begin
  if Trim(edtNome.Text) <> '' then
  begin
    lblNome.Caption := 'Preenchido';
    lblNome.Font.Color := clWebNAVY;
  end
  else
  begin
    lblNome.Caption := 'Obrigatório';
    lblNome.Font.Color := clWebRED;
  end;
end;
```

Listagem 4. Código dos botões

```
procedure TfrmCadastro.IWButton1Click(Sender: TObject);
begin
  lblNomeFinal.Caption := edtNome.Text;
  lblEnderecoFinal.Caption := edtEndereco.Text;
  lblCidadeFinal.Caption := edtCidade.Text;
  lblEstadoFinal.Caption := edtEstado.Text;
  lblCEPFinal.Caption := edtCep.Text;
  lblCursoFinal.Caption := iwcbxCursos.Items
    [iwcbxCursos.ItemIndex];
  lblConcursoFinal.Caption := iwcbxConcursos.Items
    [iwcbxConcursos.ItemIndex];
end;
```

Listagem 5. Código do evento OnAsyncChange do iwcbxCursos

```
procedure TfrmCadastro.iwcbxCursosAsyncChange(Sender:
  TObject; EventParams: TStringList);
begin
  if not sqlConexao.Connected then
    sqlConexao.Connected := True;
  sdsConcursos.Close;
  cdsConcursos.Close;
  cdsConcursos.Params.ParamByName('ID').AsInteger :=
    StrToInt(Copy(iwcbxCursos.Items[iwcbxCursos.
    ItemIndex], 1, 1));
  cdsConcursos.Open;
  if not (cdsConcursos.IsEmpty) then
  begin
    iwcbxConcursos.Items.Clear;
    while not cdsConcursos.EOF do
    begin
      iwcbxConcursos.Items.Add(cdsConcursos.FieldByName('CONCURSO').AsString);
      cdsConcursos.Next;
    end;
  end
  else
    iwcbxConcursos.Items.Clear;
  cdsConcursos.Close;
end;
```

entender de *XHTML*, *XSLT*, *XML* ou mesmo *Javascript*, isso porque a CodeGear encapsulou nos componentes todas essas tecnologias, ou seja, se digitarmos qualquer código em um destes eventos, termos um efeito Ajax. Vejamos na prática. Insira uma série de *IWLabels's* no formulário de forma que seu layout se pareça com o desenho da **Figura 7**.

Aproveite e modifique o *Name* dos *IWLabel's* vistos na **Figura 7** usando o nome que aparece no *Caption* dos mesmos. Feito isso teremos um layout semelhante ao mostrado na **Figura 8**.

Como mencionado vamos codificar nosso exemplo usando os eventos *OnAsync* dos componentes iniciando pelo *edtNome*. O evento que receberá código é o *OnAsyncExit* que acontece quando saímos do campo. Por que utilizar *OnAsyncExit*? A principal razão é que um código Ajax será executado por "baixo" da aplicação e atualizará somente o controle que desejamos, nesse caso o *lblNome* que dirá se o campo foi preenchido ou não.

Não há razão alguma em solicitar uma nova requisição ao servidor apenas para fazer o "redesenho" de um *Label* em tela. Por isso é importante o uso de Ajax em aplicações *Web*. Ganhos em agilidade são importantíssimos, principalmente em se tratando de internet de baixa velocidade.

Embora o uso de banda larga esteja aumentando gradativamente, ainda há muita gente utilizando banda estreita, ou popularmente chamada de linha discada.

Codifique o evento mencionado usando o código da **Listagem 3**. No código da **Listagem 3** apenas estamos fazendo uma verificação no *Edit*, na intenção de saber se foi ou não preenchido. Caso tenha sido, uma mensagem de "Preenchido" é mostrada em azul. Do contrário a mensagem passa a ser "Obrigatório", em vermelho. Esse recurso faz com que o usuário ganhe tempo já que não precisar apertar concluir ou aguardar que a tela seja recarregada para saber se o cadastro foi aceito.

Repita esse processo no campo *edt Endereco*. Nesse caso atualize o *Label lbl Endereco*. Execute a aplicação e preencha os dados passando de campo para campo. Perceberá que a página não é recarregada quando passamos de um campo para outro. Os *labels lblNome* e *lblEndereo* são atualizados como num passe de mágica, sem *reload*, sem *renderização*.

Façamos um novo teste. Inclua o código da **Listagem 4** no evento *OnAsyncClick* do *btnComAjax* e no *OnClick* do botão *btn SemAjax*. O código faz a atualização dos *labels* que colocamos na parte inferior da página. Execute novamente, faça o preenchimento dos campos e clique em cada um dos botões. É fácil notar a diferença.

Agora codificaremos o evento *OnAsyncChange* do primeiro *ComboBox*, *iwcbxCursos*. Ele deverá capturar o código do concurso e efetuar uma instrução *SQL* no BD retornando todos concursos ligados ao curso selecionado. Então digite o código da **Listagem 5** no evento e componente citado.

Notamos novamente que não há segredos, é código Delphi puro. Abrimos a conexão, atualizamos o parâmetro *ID* da *query* e a abrimos. Em seguida usamos seu *Result Set* para atualizar o segundo *ComboBox* com os concursos disponíveis. Usei um método de programação bem simples apenas para exemplificar. Copiamos o primeiro caractere o item selecionado em *iwcbxCursos* e o usamos como parâmetro. Porém, se tivermos um código de concurso maior que *9* a seleção não funcionará. Execute novamente a aplicação e veja o resultado.

Atualizando hora com Async

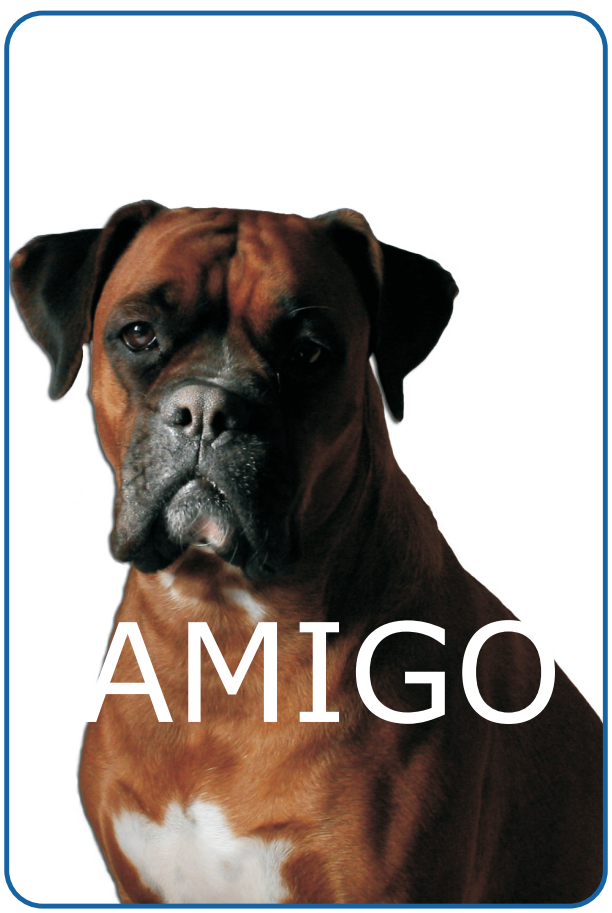
Para simplificar ainda mais o processo de entendimento do Ajax faremos um novo teste. Insira um *IWTimer* em tela e mais um *Label*("lblHora"). Digite o código a seguir no evento *OnTimer* e execute a página. Verá que a cada *1* segundo a página é recarregada atualizando o *label* com a hora atual. Desta forma, fica impossível conseguir digitar os dados de cadastro no formulário.

```
  lblHora.Caption :=
FormatDateTime('HH:NN:SS', Now);
```

Agora tente retirar o código deste evento e colocá-lo no *OnAsyncTimer*. Basta executarmos novamente a aplicação para percebermos que o *label* é atualizado individualmente, sem nenhuma requisição ao servidor. E melhor: podemos digitar nossos dados normalmente no cadastro.

Conclusão

Nesse artigo vimos que é perfeitamente possível se trabalhar com Ajax sem nem mesmo saber ao certo como ele funciona. O importante é sabermos quais são os princípios básicos de sua utilização. Aconselho incluir novos componentes em tela e tentar trabalhar com o Ajax e demais recursos da *VCL for the Web*. Boa leitura e até a próxima. ●



Existem coisas
que não conseguimos
ficar sem!

...só pra lembrar,
sua assinatura pode
estar acabando!

Renove Já!

www.devmedia.com.br/renovacao

Para mais informações:
www.devmedia.com.br/central

DevMedia GROUP

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi .NET

Diretivas de Compilação

Aprenda na prática este poderoso recurso da linguagem Delphi

Ricardo C. Boaro
(rboaro@aquasoft.com.br)
trabalha com desenvolvimento de sistemas em Delphi há mais de 10 anos e PocketStudio há 3 anos. Atualmente é gerente de informática na Di Hellen Indústria de Cosméticos, e atua como instrutor certificado Borland na Aquasoft Tecnologia da Informação parceira da Borland, em Porto Alegre – RS. Borland Instrutor, Delphi 7, 2007 e Certified.

Nesse artigo desvendaremos os "mistérios" das diretivas de compilação, para muitos considerada uma maneira complicada de desenvolver e manter os *fontes* do sistema. Antes de entrar no mérito da questão e apresentar-lhe a minha visão sobre diretivas, conceitos, utilização e por que utilizá-las, vamos entendê-las.

O que são diretivas de compilação?

As *Diretivas de Compilação* são baseadas na existência de condições para o compilador. São símbolos condicionais que trabalham como variáveis *Booleanas*, ou seja, *True* ou *False*. Qualquer condição válida é tratada como *False* até que seja definido o inverso. Podemos usar *$DEFINE* ou *$UNDEF* para especificar um conjunto de símbolos verdadeiros.

Podemos definir os blocos de códigos válidos identificando as condições válidas em *Project>Options*, na aba *Directories/Conditionals* no item *Conditional defines*.

As diretivas *$IFDEF*, *$IFNDEF*, *$IF*, *$ELSEIF*, *$ELSE*, *$ENDIF*, e *$IFEND*, permitem ao compilador omitir blocos de código, baseado no status da condição *$IF* e *$ELSEIF*. Algumas características das diretivas são listadas a seguir:

- Podem ser aninhadas, em até 32 níveis;
- Seu identificador deve iniciar com letras seguidas por uma combinação de letras, números e *underscores*;
- Podem ter qualquer tamanho, mas apenas os primeiros 255 caracteres serão considerados;

Logicamente que não teremos um identificador maior que 255 caracteres. O Delphi utiliza diretivas de compilação em seu fonte para identificar muitas variáveis, na **Listagem 1** segue alguns exemplos da utilização no próprio Delphi.

A **Listagem 1** é o início da declaração da *Unit classes* do Delphi. Perceba que após a seção *Interface* existem duas diretivas

identificando quais *Units* serão utilizadas na unidade *Classes* de acordo com o sistema operacional.

Nesse exemplo estão sendo utilizadas as diretivas *{$IFDEF Condição}* e *{$ENDIF}*, elas são as mais comuns e fáceis de utilizarmos por serem análogas ao *IF..Then*. O Delphi utiliza diretivas para identificar a sua versão da mesma forma que as utiliza para identificar se estamos desenvolvendo uma aplicação WIN32 ou .NET.

Existem muitos tipos de diretivas e precisaria muito mais que um artigo para estudarmos cada uma delas, e qual a sua funcionalidade. Ao invés disso criaremos um exemplo para abstrair da nossa realidade a aplicabilidade.

Onde podemos utilizá-las?

Imagine a seguinte situação: Possuímos um ERP completíssimo, com um cadastro de clientes excepcional, controle de estoque perfeito, financeiro totalmente integrado à contabilidade, enfim para nós ele é o ERP perfeito. Mas, sempre haverá um cliente querendo alterá-lo e adaptá-lo a sua empresa, e mais, cada cliente terá a sua particularidade. Por mais que tenhamos um sistema "completo", as empresas nos mais variados ramos, trabalham da forma mais variada possível, jamais existirá uma empresa com a mesma maneira de trabalhar que outra.

Cada uma vive a sua realidade, sendo assim, tem sua necessidade específica. Como resolver essa situação? Manter vários sistemas? Fazer a famosa POG, Programação Orientada a Gambiarra? Claro que não, vários sistemas seriam difíceis de manter e elevariam muito o custo de manutenção. POG está descartada, pois o sinônimo de POG com certeza são problemas futuros. Os famosos remendos podem resolver uma determinada situação momentaneamente, mas com o passar do tempo irão causar novos problemas e isso acabará sendo uma bola de neve até que o sistema se transforme em uma fonte de estresse ilimitada.

Sendo assim meus amigos, vamos nos valer das muitas facilidades oferecidas pelo Delphi para facilitarmos nosso trabalho e melhorarmos a qualidade dos nossos sistemas. Nessa situação Diretivas de Compilação serão a nossa salvação, pois são fáceis de implementar e fáceis de manter.

Vamos criar um exemplo para fixarmos a utilização das mesmas. As diretivas estão disponíveis em todas as versões do Delphi em nosso exemplo trabalharemos com o Delphi 7 criando uma aplicação para Win32.

Inicie uma nova aplicação em *File|New|Application*. A idéia inicial é criarmos um exemplo simples, mas que será muito útil para entendermos o que podemos alterar trabalhando com diretivas.

Digamos que temos o *Cliente A* que possui uma base de dados Paradox, pois é uma empresa pequena e trabalha apenas com base de dados local. Desenvolvemos para ele um cadastro de clientes simples apenas com os dados necessários para que ele possa consultar suas informações quando precisar entrar em contato com o mesmo, sendo assim utilizamos os componentes da paleta BDE.

Por outro lado temos o *Cliente B* que possui uma empresa considerável que usa os dados dos clientes para emitir notas fiscais e controlar seu limite de compra. Para complicar um pouco mais, ele tem vários terminais acessando o cadastro de clientes ao mesmo tempo, consultando, utilizando os dados para emitir notas fiscais etc. Nesse cenário precisamos trabalhar com uma base de dados mais robusta, e melhor preparada para o uso em rede. Para isso utilizaremos o SQL Server Express, e acessaremos a base de dados utilizando os componentes da paleta *dgGo*.

Listagem 1. Exemplo de diretivas de compilação

```
unit Classes;
{$R-,T-,X+,H+,B-}
{$IFDEF MSWINDOWS}
{ ACTIVEX.HPP is not required by CLASSES.HPP }
(*$NOINCLUDE ActiveX*)
{$ENDIF}
{$IFDEF LINUX}
{$DEFINE _WIN32}
{$ENDIF}
{$IFDEF MSWINDOWS}
{$DEFINE _WIN32}
{$ENDIF}

interface
{$IFDEF MSWINDOWS}
  uses Windows, Messages, SysUtils, Variants, TypInfo, ActiveX;
{$ENDIF}
{$IFDEF LINUX}
  uses Libc, SysUtils, Variants, TypInfo, Types;
{$ENDIF}
```

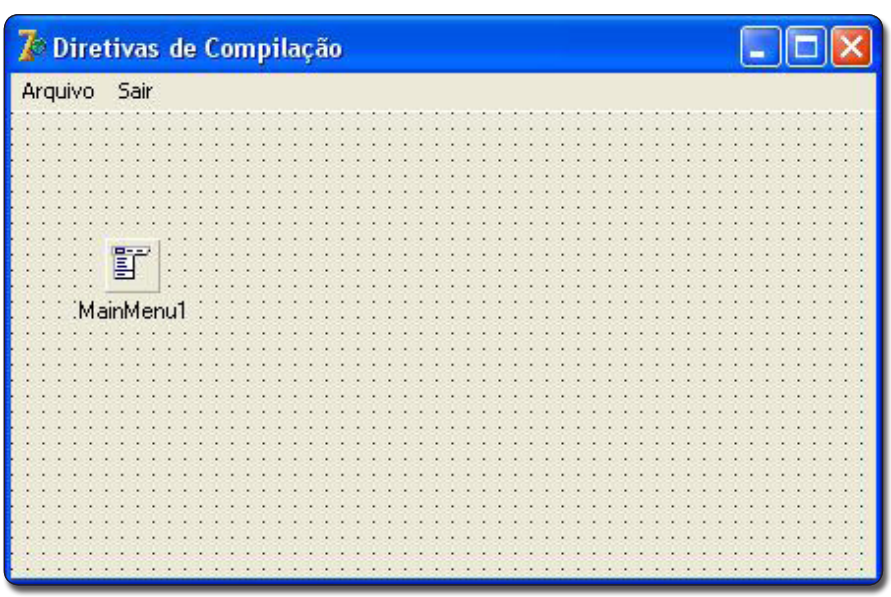


Figura 1. Exemplo de tela principal

Construindo o Exemplo

Vamos nos basear no exemplo exposto na explicação acima, temos o ClienteA, pequeno com um sistema de cadastro de clientes mono-usuário, que irá funcionar apenas no servidor. Já o ClienteB possui uma estrutura robusta e tem vários acessos a base de dados simultaneamente. Partindo desse princípio vamos criar uma tela principal com um componente *MainMenu* e configurá-lo como mostrado na **Figura 1**.

Após a configuração do componente *MainMenu*, vamos adicionar um novo formulário em nosso projeto que será o cadastro de clientes. Devemos salvá-lo como "UClientes.pas", e na propriedade *Name* chamá-lo de "FrmClientes". Seu *Caption* deve ser configurado com "Cadastro de Clientes". Estou partindo do princípio que já temos a base de dados configurada, não vou entrar em detalhes aqui sobre a criação do BD no SQL Server e no Paradox.

Para manipularmos as tabelas que possuem os mesmos campos, montei essa estrutura propositalmente para que possamos simplificar o exemplo e facilitar o entendimento do exercício proposto. A tela de clientes deve ser configurada com *DBEdit's* e *Button's* como mostrado na **Figura 2**. O código completo dos botões superiores você encontra **Listagem 2**.

Feito isso vamos adicionar um Data Module ao nosso projeto e salvá-lo com o nome de "uDmBase.pas" e nomeá-lo como "DmBase". Nele iremos adicionar um componente *Table*("TblClientes") da paleta *BDE*, que será ligado a nossa tabela clientes do *Paradox* através de um *alias*. Um componente *ADOConnection*, que deve ser conectado ao servidor SQL Server, e um *ADOTable*("AdoTblClientes") ligado ao *AdoConnection* e na tabela de Clientes do SQL Server. Na **Figura 3** temos o *DmBase* configurado.

Feito isso temos a estrutura do nosso programa montada. Lembrem-se que o *ClienteA* deve trabalhar com a tabela de clientes ligada ao Paradox e o *ClienteB* com a tabela de clientes ligado ao SQL Server. A idéia é utilizarmos a mesma tela de cadastro de clientes para os dois clientes, mudando apenas a base que será utilizada conforme a diretiva informada.

Vamos colocar em prática nossa idéia. Percebam na **Listagem 1** que estou chamando o *Insert, Edit* etc., diretamente de um *DataSource*. Esse *DataSource*("dtClientes") deve ser adicionado ao formulário de clientes.

Estamos chamando os métodos de inclusão, alteração etc. diretamente da propriedade *DataSet*, assim não precisamos saber se o *DataSource* está conectado ao Paradox ou ao SQL Server. A única coisa que precisamos fazer é mudar seu *DataSet* para apontar para uma base ou outra conforme a compilação que fizermos, utilizando a diretiva do *ClienteA* ou do *ClienteB*.

Para informarmos ao compilador que utilizaremos uma diretiva ou outra vamos em *Project>Options*, na aba *Directories/Conditionals* e logo após em *Conditional defines*. Como explicado no

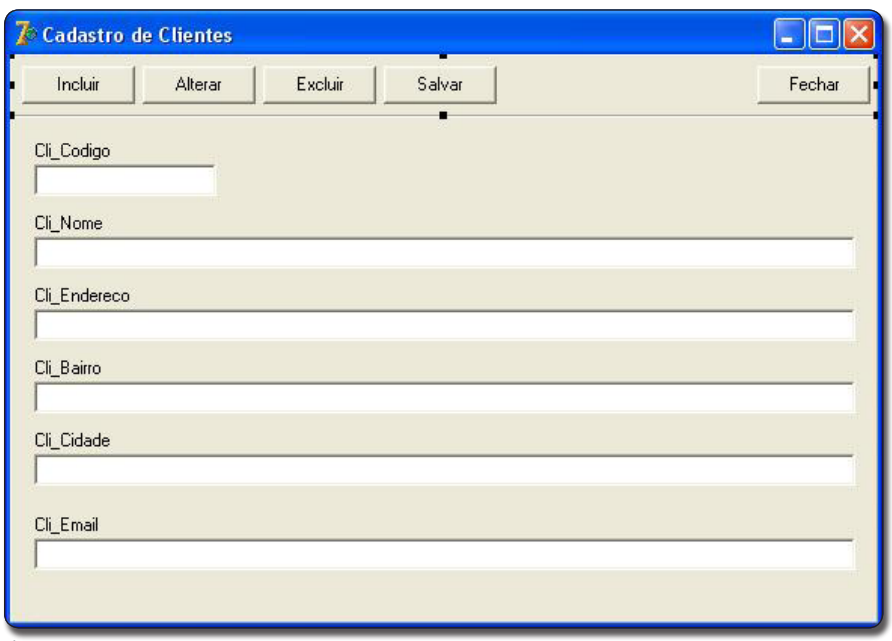


Figura 2. Exemplo de tela de clientes

Listagem 2. Código dos botões

```
procedure TFrmClientes.BtnInserirClick(Sender: TObject);
begin
  DtClientes.DataSet.Insert;
end;
procedure TFrmClientes.BtnAlterarClick(Sender: TObject);
begin
  DtClientes.DataSet.Edit;
end;
procedure TFrmClientes.BtnExcluirClick(Sender: TObject);
begin
  DtClientes.DataSet.Delete;
end;
procedure TFrmClientes.BtnSalvarClick(Sender: TObject);
begin
  DtClientes.DataSet.Post;
end;
procedure TFrmClientes.BtnFecharClick(Sender: TObject);
begin
  Close;
end;
```

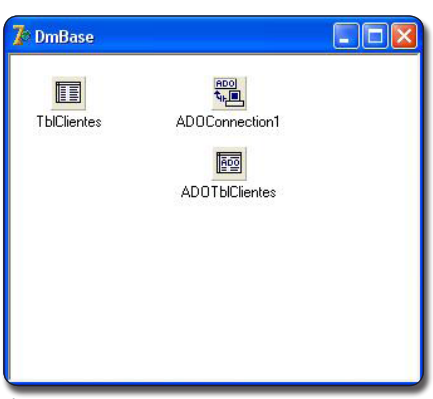


Figura 3. Data Module já configurado

início do artigo para iniciarmos vamos definir a diretiva *ClienteA* para ficar bem intuitivo, como mostra a **Figura 4**.

O próximo passo é informarmos no código do projeto como ele irá tratar as condições, por isso codifique o evento *OnCreate* do formulário de clientes conforme a **Listagem 3**.

Note que estou utilizando as duas diretivas que vamos trabalhar em nosso exemplo e vejam que é simples para trocarmos de uma base para outra. Para incrementar um pouco mais nosso exemplo no evento *OnCreate* do formulário principal, vamos digitar o código da **Listagem 4.**

Vamos rodar a aplicação com a diretiva do *ClienteA* ativa e ver o resultado. Logo no formulário principal podemos ver a mudança. Como mostrado na **Figura 5** o fundo do formulário foi modificado para a cor azul. Feche a aplicação, modifique a diretiva para *ClienteB* e execute novamente o sistema. Note que novamente a cor é modificada na **Figura 6**.

Logicamente aqui percebemos apenas as alterações na cor e no *Caption* do formulário, mas acessando o cadastro de clientes podemos incluir registros tanto para o *ClienteA* acessando o Paradox como para o *ClienteB* acessando o SQL Server. O importante é que você entenda que podemos alterar qualquer configuração no projeto trabalhando com diretivas.

Considerações Importantes

As diretivas de compilação só serão atualizadas quando trocarmos da diretiva *ClienteA* para *ClienteB* fazendo um *Buid* da aplicação. Se apenas compilarmos, a diretiva não é aceita. Um dos segredos da compilação é sempre

Listagem 3. Código do evento OnCreate do formulário

```
procedure TFrmClientes.FormCreate(Sender: TObject);
begin
 {$IFDEF ClienteA}
  DtClientes.DataSet := DmBase.TblClientes;
 {$EndIF}
 {$IFDEF ClienteB}
  DtClientes.DataSet := DmBase.ADOtblClientes;
 {$EndIF}
end;
```

Listagem 4. Evento OnCreate do formulário principal

```
procedure TFrmDiretivas.FormCreate(Sender: TObject);
begin
 {$IFDEF ClienteA}
   Application.Title:= 'Cliente A';
   FrmDiretivas.Caption:= 'Sistema de controle de clientes do Cliente A';
   FrmDiretivas.Color:= ClBlue;
 {$EndIF}
 {$IFDEF ClienteB}
   Application.Title:= 'Cliente B';
   FrmDiretivas.Caption:= 'Sistema de controle de clientes do Cliente B';
   FrmDiretivas.Color:= ClRed;
 {$EndIF}
end;
```

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

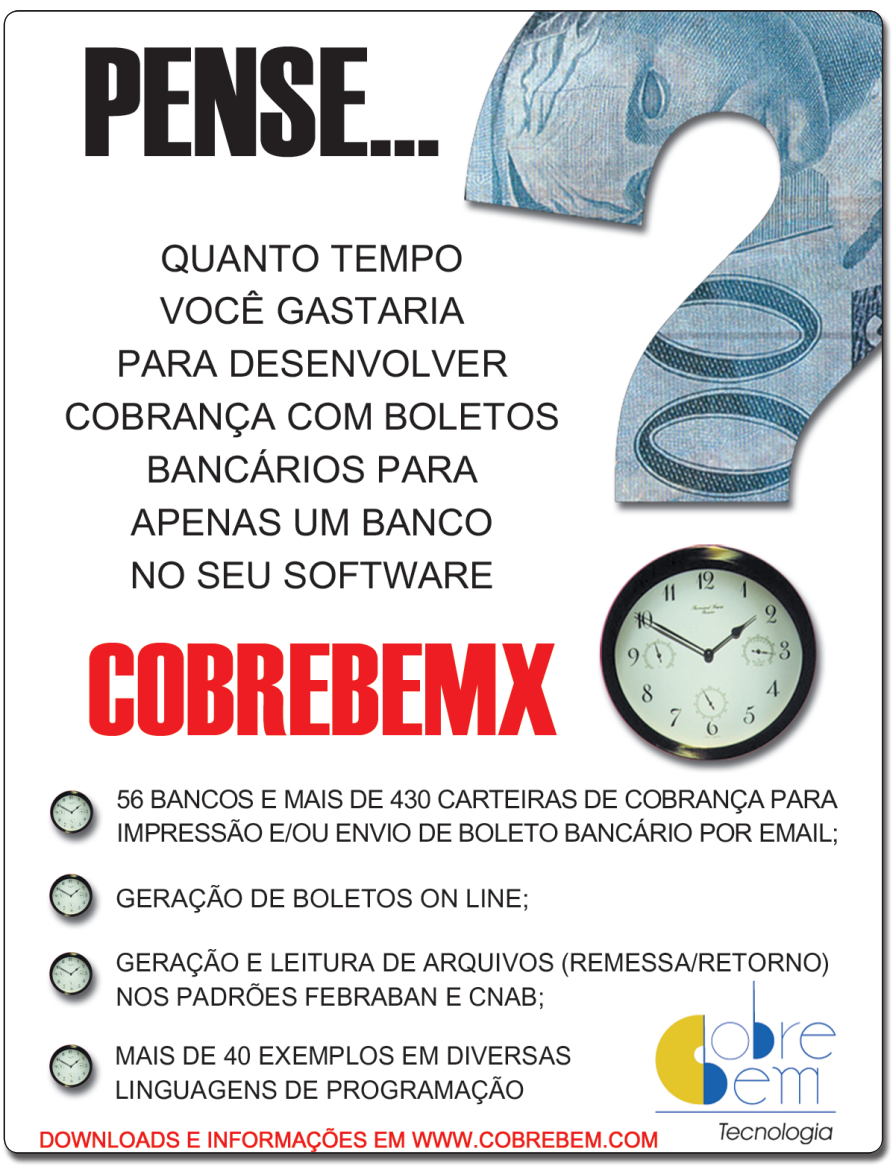
Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Ricardo Boaro que mostra como trabalhar com diretivas de compilação.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=7248&hl=diretivas

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli que mostra como trabalhar com diretivas de compilação.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=530&hl=diretivas



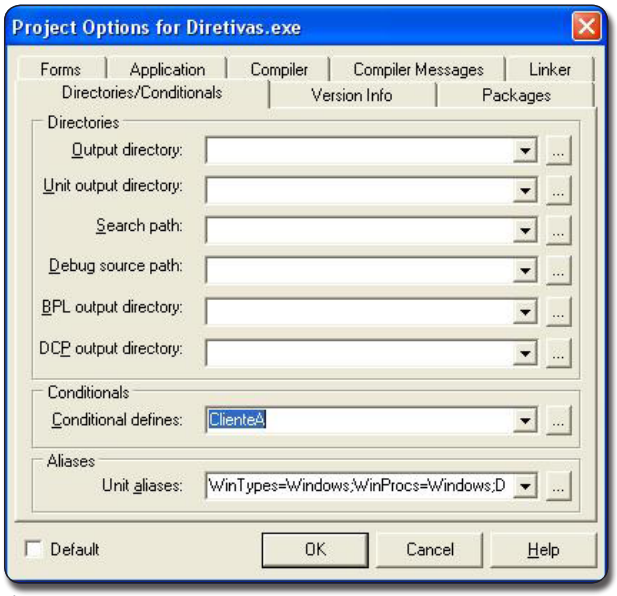
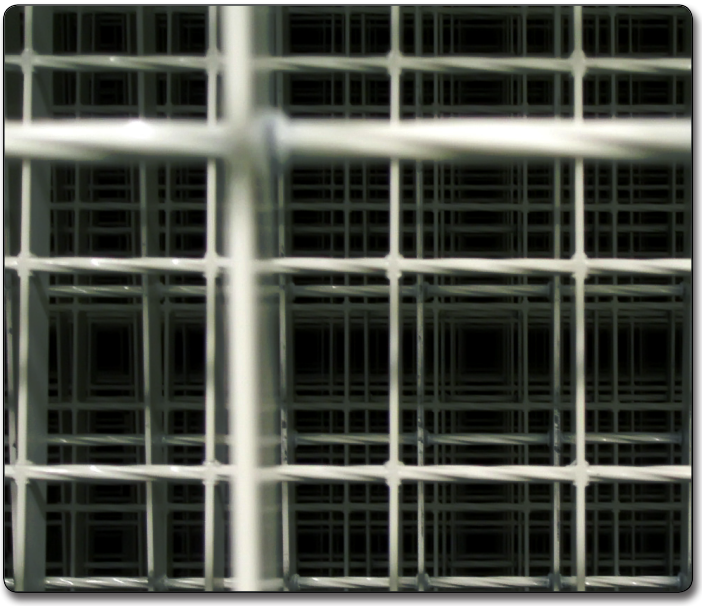


Figura 4. Definindo a diretiva de compilação



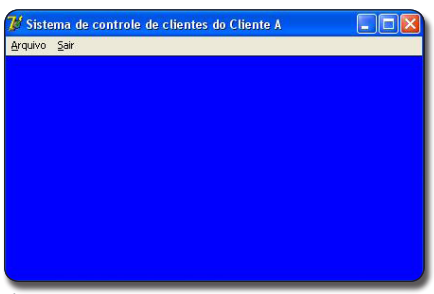


Figura 5. Fomulário principal em azul devido à diretiva ClienteA

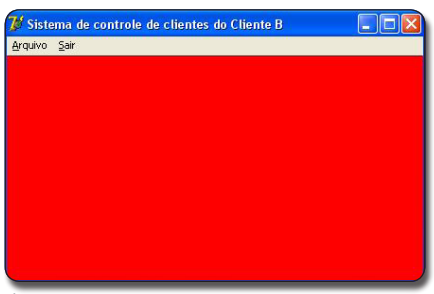


Figura 6. Fomulário principal em vermelho devido à diretiva ClienteB

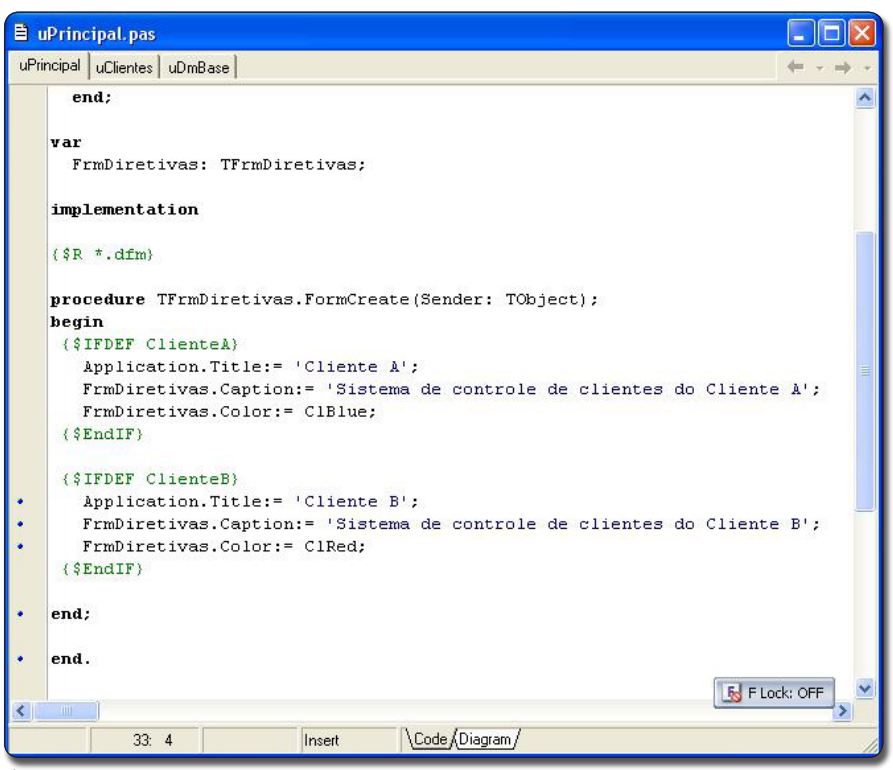


Figura 7. Indicação de fluxo de código

trabalharmos com *Build* ao final de cada alteração de condição.

Outro detalhe importante sobre diretivas é entendermos que os códigos que estão dentro das diretivas que não fazem parte da condição, ou seja, quando definimos que a diretiva de compilação ativa é a *ClienteA*, todos os códigos que estiverem dentro da condição *ClienteB* serão ignorados, e não serão carregados no executável.

Ao compilarmos podemos notar, pelos indicadores azuis, que o fluxo do programa não passará pelo código referente ao *ClienteA*. (ver **Figura 7**)

Dessa forma mesmo que houvesse um erro de sintaxe dentro da diretiva *ClienteA* não seria notado, nem mesmo o *Code Completion* funciona bem dentro de seções de diretivas desativadas.

Conclusão

Neste artigo entendemos o funcionamento básico das diretivas de compilação. É importante aproveitarmos ao máximo os recursos da ferramenta para podermos ganhar o máximo em produtividade e redução dos custos de nossos softwares.

O exercício criado foi simples, mas acredito que foi o suficiente para começarmos a trabalhar com diretivas. Feito isso libertem a sua imaginação e tirem proveito de todas as formas imagináveis. Um grande abraço a todos e bons códigos. ●

Cursos Online

Assinatura

ClubeDelphi PLUS

Mais conteúdo DELPHI por muito menos!

A Revista **ClubeDelphi** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade . Conheça abaixo os cursos já disponíveis..

Curso em destaque

Aplicações Client/Server com dbExpress e Firebird

Confira neste curso online como criar uma aplicação client/server completa no Delphi 7, utilizando dbExpress e Firebird.
Aprenda também: Como trabalhar com um driver dbExpress específico e gratuito para o Firebird ; Como utilizar Orientação a Objetos, e técnicas como herança visual de formulários e relatórios ; saiba como colocar regras de negócios no banco de dados, usando Triggers e Stored Procedures; E crie relatórios com o Quick Report, Rave Reports e Report Builder, e muito mais.

Confira o plano de aula completo:
www.devmedia.com.br/clienteserver

A sua melhor opção de aprendizagem!
Assine a **ClubeDelphi** e comece já seu treinamento!
www.devmedia.com.br/assine

Outros cursos disponíveis: **www.devmedia.com.br/curso**

- Curso Online - Sistema SysCash
- Aplicações client/server com Windows Forms no Delphi 2006
- Criando uma Aplicação multi-camadas Completa com Delphi
- Aplicações WEB com IntraWeb e Delphi 7 (Delphi Win32)

DevMedia GROUP **Mais Informações: www.devmedia.com.br/central - Tel.: 21 2220-5375 / 2220-5435**

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Dicas de QuickReport

Conheça algumas dicas para facilitar seus relatórios com QuickReport

Maikel Marcelo Scheid
(maikelscheid@gmail.com)
é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MSSQL. É membro da Equipe Editorial ClubeDelphi.

As ferramentas de relatórios são um meio do programador proporcionar aos usuários formas mais amplas e práticas de visualizar informações armazenadas em uma base de dados. Informações estas que podem ser oriundas de telas de cadastros ou de ações geradas pelo próprio sistema, como cadastros de cidades, clientes, produtos ou até mesmo o registro de *log* criado por um usuário ao executar alguma ação no sistema.

O QuickReport acompanha o Delphi desde sua primeira versão e até a versão 6 vinha incorporado ao seu IDE. Na versão do Delphi 7 e posteriores, com o surgimento do Rave Reports, o Quick Report deixou de vir pré-instalado no IDE, ou seja, é necessário fazer a instalação manual do componente. Já no BDS 2006 e RAD Studio 2007 é necessário fazer o *download* e instalação a partir do site da CodeGear (somente usuários registrados).

Neste artigo veremos a instalação dos componentes do QuickReport no Delphi 7 e criaremos relatórios dinâmicos através de formulários e listagem de informações resgatadas do banco de dados. Passaremos a aplicar filtros de pesquisa em tempo de execução. Veremos também uma forma prática de customizar o layout dos relatórios criados com o QuickReport. Aplicaremos regras de formatação e também uma forma simples de agrupar registros.

> **Nota**: Para consultar mais detalhes, novidades, informações técnicas, download e/ou suporte técnico do pacote QuickReport consulte o site do fabricante em *www.qusoft.com*.

Instalando o pacote de componentes

Durante este artigo iremos utilizar o Delphi 7 para criação da aplicação e exemplos práticos para o uso de

relatórios com os componentes do QuickReport. Como abordado na introdução ao artigo, na versão do Delphi 7 e posteriores o QuickReport deixou de vir pré-instalado no IDE. Para tê-lo instalado será necessário adicionar sua BPL ao Delphi.

Feche todos os formulários abertos o menu *File|Close All* e inicie o processo de instalação no menu *Component>Install Packages*. Clique no botão *Add* e navegue até a pasta de *Bin* do Delphi que por *default* é *C:\Arquivos de Programas\Borland\Delphi7\Bin*. Localize o arquivo *dclqrt70.bpl* e selecione-o. Feito isso o pacote de componente já é mostrado na lista como *QuickReport Components* conforme podemos ver na **Figura 1**.

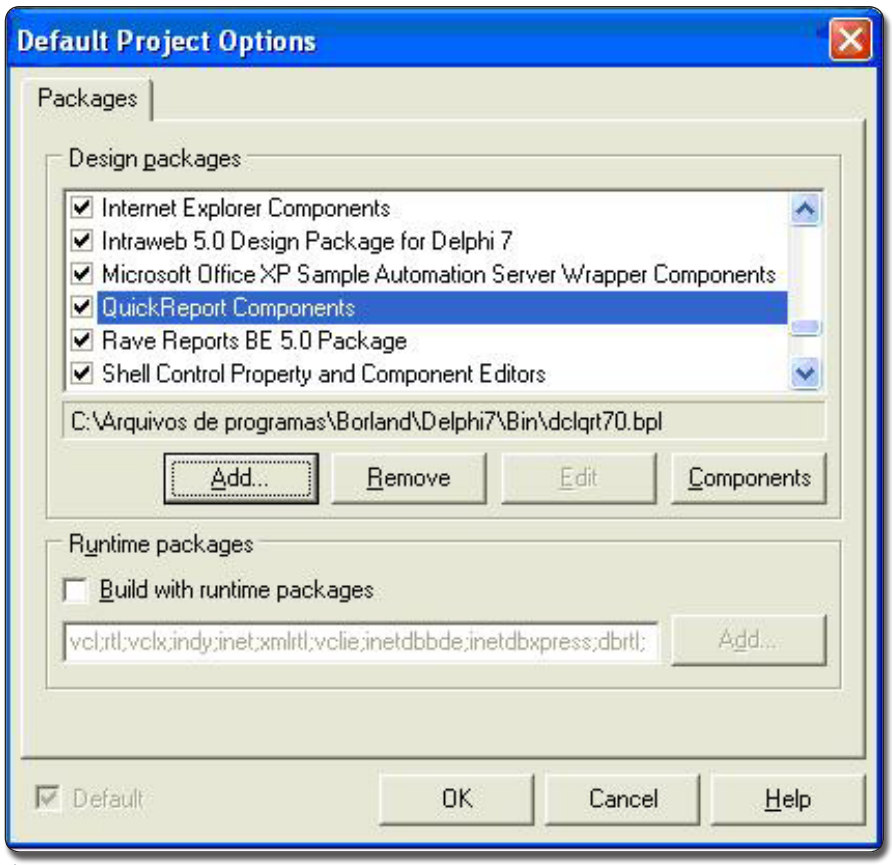


Figura 1. Janela de instalação de pacotes

Criando a aplicação

Para demonstração dos relatórios criaremos uma aplicação com acesso ao banco de dados *Employee.fdb* que acompanha a instalação do Firebird e a partir deste iremos configurar nossas situações de relatório por meio de pesquisas e filtros realizadas nos componentes.

No menu *File|New>Application* crie uma nova aplicação e altere a propriedade *Name* do formulário inicial para "frmPrincipal" e a propriedade *Caption* para "Dicas QuickReport". Salve a *Unit* principal do projeto como "uPrincipal.pas" e o projeto salve como "prjQuickReport.dpr".

Para criar a conexão com banco de dados e estruturar os componentes, vamos utilizar um *Data Module* que deverá ser criado através do menu *File|New>Data Module*. Altere seu nome para "DM" e salve a *Unit* como "uDM.pas". Adicione um componente *SQLConnection*("sqlConexao") da paleta *dbExpress*.

O *sqlConexao* é o responsável pela conexão com o arquivo do banco de dados, em nosso caso será o arquivo *Employee.fdb*, normalmente situado em *C:\Arquivos de Programas\Firebird\Firebird(versão)\examples*. Para configurar os parâmetros de acesso ao banco de dados, com um duplo clique sobre o componente *sqlConexao*, crie uma nova conexão utilizando o ícone *Add Connection* e defina o *Driver Name* para *Interbase* e *Connection Name* para "Employee" (**Figura 2**). Confirme e prossiga para a configuração dos demais parâmetros, começando pela propriedade *Database* onde deverá ser informado o caminho da base de dados *Employee.fdb*.

Figura 2. Criando uma nova conexão

Defina o usuário do Firebird (SYSDBA) e a senha (*masterkey*) para acesso ao BD. Na propriedade *ServerCharSet* utilizaremos o padrão *Win1252* e em *SQLDialect* utilize o padrão *3* para que possamos utilizar comandos de *SQL* mais complexos (**Figura 3**).

Ainda no componente *sqlConexao*, defina sua propriedade *LoginPrompt* para *False* e altere a propriedade *Connected* para *True* testando assim o sucesso da conexão.

Adicione também no *Data Module* um componente *SQLDataSet*("sqlPesquisa"), relacione a propriedade *SQLConnection* ao componente *sqlConexao* e digite a instrução *SQL* da **Listagem 1** em sua propriedade *CommandText*.

O comando de *SQL* que adicionamos irá trazer como resultado todos os registros de empregados contidos na tabela *EMPLOYEE* e relacionando-os a

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp
Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Adriano Santos que mostra como instalar e utilizar o editor de relatórios QuickReport.
www.devmedia.com.br/articles/viewcomp.asp?comp=6588

seus respectivos departamentos através de um *Inner Join* com a tabela *DEPARTAMENT*. O objetivo de selecionar os empregados com seus respectivos departamentos é criar logo mais um relatório onde serão criados grupos de empregados de acordo com o departamento onde cada um atua.

Adicione ainda ao *Data Module* um componente *DataSetProvider*("dspPesquisa") da paleta *Data Access* relacionando sua propriedade *DataSet* para *sqlPesquisa*. Adicione ainda da paleta *Data Access* um componente *ClientDataSet*("cdsPesquisa") e configure a ligação da propriedade *ProviderName* ao *dspPesquisa*. Com um duplo clique sobre o *cdsPesquisa* abra o *Fields Editor* e adicione todos os campos da tabela através do clique de direita e *Add all fields*.

Retornando ao formulário principal, adicione três componentes *Button* da paleta *Standard* alterando sua propriedade *Caption* para "Geral", "Agrupado" e "Filtrar", respectivamente. Adicione ainda um componente *Edit*("edtFiltro") e exclua o valor em sua propriedade *Text*. Adicione também um *Label* e altere seu *Caption* para "Pesquisa".

Adicione um componente *DataSource* ("dsEmpregados") relacionando sua propriedade *DataSet* ao *cdsPesquisa* configurado no *DM*. Adicione um componente *DBGrid*("gridEmpregados") e ligue a propriedade *DataSource* ao *dsEmpregados*. Organizando os componentes no formulário de forma semelhante à **Figura 4.**

Adicione ao evento *OnShow* do *frmPrincipal* a seguinte linha de comando que fará a abertura do *ClientDataSet* de forma a exibir todos os registros contidos na base de dados:

```
dsEmpregados.DataSet.Open;
```

Adicione também ao evento *OnClose* do *frmPrincipal* o seguinte código que irá fechar a conexão de pesquisa com a tabela da base de dados:

```
dsEmpregados.DataSet.Close;
```

Listagem 1. Instrução SQL do sqlPesquisa

```
SELECT
  EMPLOYEE.FULL_NAME, EMPLOYEE.PHONE_EXT,
  EMPLOYEE.JOB_COUNTRY, EMPLOYEE.SALARY, DEPARTMENT.DEPARTMENT
FROM
  EMPLOYEE
INNER JOIN DEPARTMENT
ON(EMPLOYEE.DEPT_NO = DEPARTMENT.DEPT_NO)
```

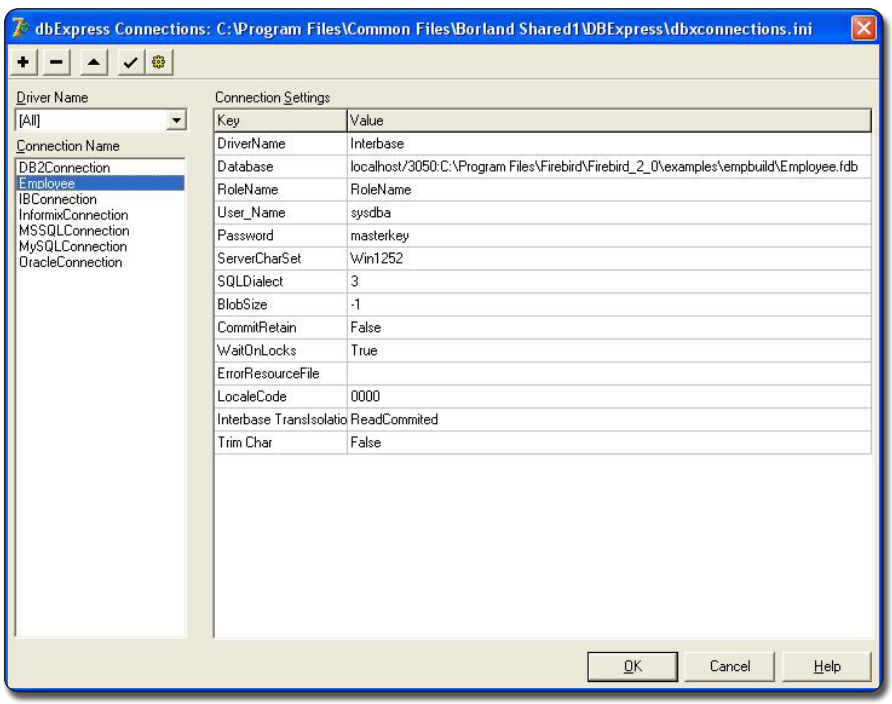


Figura 3. Configurando a conexão

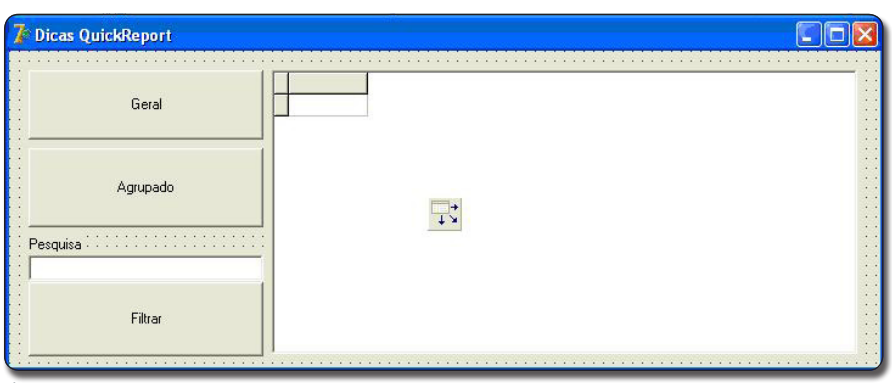


Figura 4. Layout do formulário principal

Criando os relatórios

A criação de relatórios através do QuickReport poderá ser feita por meio de duas seqüências dos processos, uma onde criamos o formulário e adicionamos todos os componentes do *QR* manualmente e outra onde criamos um novo *Report* que traz uma estrutura de relatório previamente configurada. Veremos as duas maneiras começando pela criação de um relatório geral de amostragem dos registros. O desenvolvimento será totalmente manual.

Crie, através do menu *File|New>Form*, um novo formulário("frmReportGeral") alterando o *Caption* para "Relatório Geral de Empregados" e sua propriedade *KeyPreview* para *True* e salve sua *Unit* como "uReportGeral.pas". Adicione ao evento *OnKeyDown* do *frmReportGeral* o seguinte código com a função de fechar a tela de exibição ao pressionarmos a tecla *ESC*.

```
if Key = VK_ESCAPE then
  Close;
```

Insira um componente *QuickRep* ("QRelatorio") da paleta *QReport*. Note que na área em branco do componente

que acabamos de adicionar existe uma margem em torno de todo o documento, margem esta que delimita o espaço útil do relatório, ou seja, é a área de impressão que podemos utilizar. Por *default* o relatório vem sem nenhuma *banda*. As bandas são áreas no relatório onde incluímos os componentes de visualização dos dados.

Para adicionar bandas no relatório, ou seja, espaços divididos para a exibição de cabeçalho, descrição, resultados, entre outros precisamos adicionar ao nosso formulário componentes *QRBand* e configurá-los de acordo com a função que os mesmos deverão assumir.

Adicionando um componente *QRBand* ("bdCabecalho"), que por *Default* é adicionado como *rbTitle* em *BandType* observe que o mesmo já tomou posição ao topo do relatório, onde deverá permanecer e será configurado para que este seja o cabeçalho do relatório. Altere o *BandType* para *rbPageHeader*. Arraste para dentro do *bdCabecalho* um componente *QRLabel* alterando seu *Caption* para "Relatório Geral de Empregados" e definindo um tamanho de fonte diferenciado para que assuma as características de título do relatório.

Para configurar os itens de descrição dos campos que serão exibidos no relatório adicione novamente ao formulário um componente *QRBand*("bdTitulos"), onde deverão ser adicionados quatro componentes *QRLabel* com as seguintes descrições na propriedade *Caption*: "Empregado", "Telefone", "Salário" e "Departamento". Configure a altura da *bdTitulos* para que o tamanho da linha fique ajustado de forma correta em relação ao tamanho dos títulos de colunas e altere sua propriedade *BandType* para *rbTitle*.

Para dar mais destaque à linha de descrição de cada coluna, selecione a *bdTitulos* e altere sua propriedade *Color* para *clSilver*. Da mesma forma altere também a cor de fundo dos componentes *QRLabel* também para *clSilver*. O próximo passo na criação do relatório será adicionar uma nova *QRBand* ("bdCampos") onde deverão ser exibidos os registros através do componente de texto *QRDBText* da paleta *QReport*.

Seguindo o mesmo esquema do cabeçalho das colunas, adicione na *bdCampos*, imediatamente abaixo os campos do cabeçalho, quatro componentes *QRDBText*, observando uma configuração individual para cada um deles que deverá ser de atribuir sua propriedade *DataSet* ao *cdsPesquisa* configurado no *Data Module* e também de configurar qual será o campo ao qual o *QRDBText* irá se relacionar, alterando sua propriedade *DataField* ao campo respectivo com a descrição do texto no cabeçalho.

> **Nota**: Não se esqueça de acrescentar a *Unit* do *Data Module* ao *Uses* do formulário de relatório. Use a opção *File>Use unit* ou *Alt + F11*

Ainda antes de executar seu relatório, selecione o componente *QuickRep* ("QRelatorio") e ligue a propriedade *DataSet* ao componentes *cdsPesquisa* também configurado no *Data Module* da nossa aplicação.

Depois da organização dos componentes, seu relatório deverá ficar semelhante ao relatório ilustrado na **Figura 5**. Para executar o relatório, clique com o botão direito do mouse sobre o relatório e selecione a opção *Preview*, que irá exibir as informações contidas no *ClientDataSet* configurado. Tome o cuidado de ativar o *cdsPesquisa* antes da visualização dos registros.

Para executar a tela do relatório em *Runtime*, retorne ao *frmPrincipal* onde deverá atribuir a seguinte linha de código ao evento *OnClick* do botão *Geral*.

```
Application.CreateForm(TfrmReportGeral,frmR
eportGeral);
DM.cdsPesquisa.Open;
frmReportGeral.QRelatorio.Preview;
```

O código cria o formulário, abre o *ClientDataSet* que contém os dados e por fim exibe o relatório. Vá também até o menu *Project|Options* do Delphi e no configurador de opções que irá aparecer localize a aba *Forms*. Veja que existem, relacionados na seção *Auto-Create forms*, o *Data Module* e também os dois formulários que criamos. Arraste o *frmReportGeral* para a seção *Available forms* e também arraste o *DM* para que seja criado antes que o formulário principal do sistema (**Figura 6**).

Aplicando filtros ao relatório geral de empregados

A opção de aplicar filtros ao relatório nada mais é do que uma simples configuração que faremos da propriedade *Filter* do *cdsPesquisa*, mas preferi abordar este detalhe aqui no artigo para demonstrar diferentes situações possíveis de customização e exibição do seu relatório.

Posicionado no *frmPrincipal*, adicione ao evento *OnClick* do botão *Filtrar* o código da **Listagem 2** que fará uma varredura em todos os registros do campo *FULL_NAME* procurando pelo valor digitado no espaço da pesquisa.

Ao executar sua aplicação, digite um valor no campo de pesquisa e aplique o botão *Filtrar* para visualizar os resultados.

Figura 5. Layout do relatório geral

Listagem 2. Aplicando filtros aos registros exibidos no relatório

```
procedure TfrmPrincipal.Button3Click(Sender: TObject);
begin
  Application.CreateForm(TfrmReportGeral,frmReportGeral);
  DM.cdsPesquisa.Open;
  DM.cdsPesquisa.Filtered := False;
  DM.cdsPesquisa.Filter   := 'UPPER(FULL_NAME) LIKE '+
    QuotedStr('%'+UpperCase(edtFiltro.Text)+'%');
  DM.cdsPesquisa.Filtered := True ;
  frmReportGeral.QRelatorio.Preview;
end;
```

Listagem 3. Instrução SQL para impressão de relatórios por grupo

```
SELECT
  EMPLOYEE.FULL_NAME, EMPLOYEE.PHONE_EXT,
  EMPLOYEE.JOB_COUNTRY, EMPLOYEE.SALARY,
  DEPARTMENT.DEPARTMENT
FROM
  EMPLOYEE
INNER JOIN DEPARTMENT
ON (EMPLOYEE.DEPT_NO = DEPARTMENT.DEPT_NO)
```

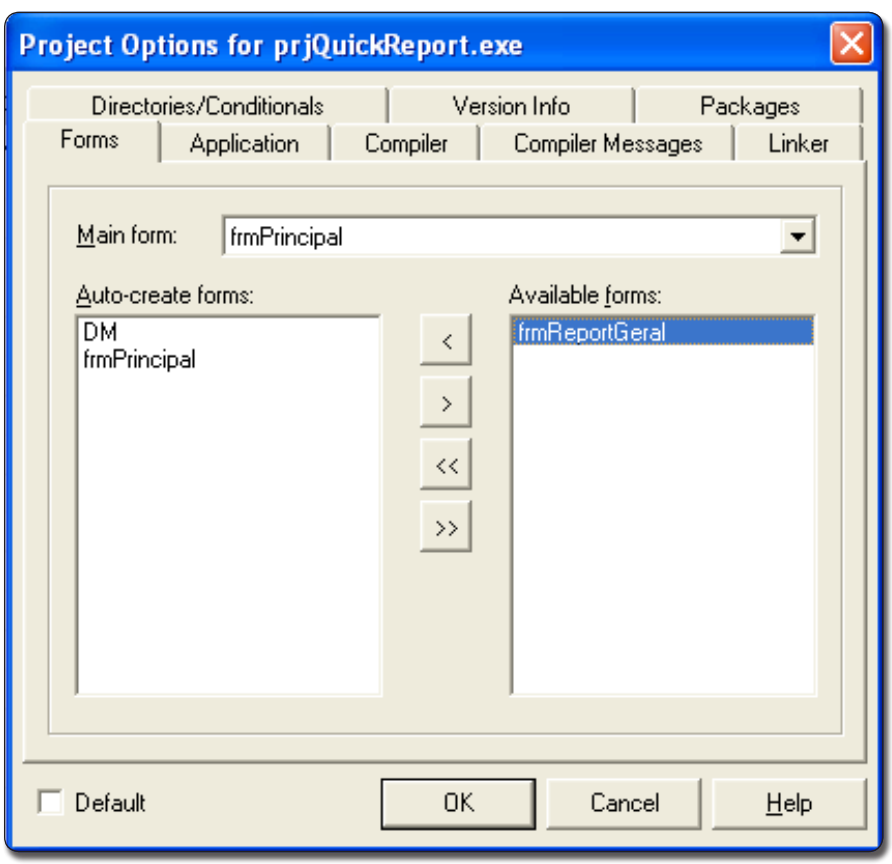


Figura 6. Opções do projeto

Ao pesquisar, por exemplo, pelo valor "be" serão listados somente alguns registros que contenham estas siglas no nome. Uma opção interessante que podemos usar na hora de exibir o relatório, já que a estrutura de componentes utilizados para ambos os exemplos anteriores é a mesma, será utilizar um componente *RadioGroup* e definir as opções para que o usuário escolha entre um relatório completo ("sem filtros") ou um relatório customizado, onde poderá antes filtrar pelos registros que realmente deseja que sejam impressos.

Criando relatório com grupos de registros

Nesta seção do artigo veremos uma opção interessante para criação de um relatório, utilizaremos basicamente as informações semelhantes ao exemplo anterior na qual agora iremos organizar os registros por grupos, ou seja, iremos criar uma lista de departamentos e logo abaixo de cada departamento uma listagem de todos os empregados.

Selecione *File|New|Other>New>Report*. Esta é uma outra forma de se criar relatórios no Delphi utilizando o Quick Report. Salve a *Unit* do formulário como "uGroupReport.pas". Adicione, da paleta *QReport*, um componente *QRGroup* ao relatório e adicione também um componente *QRBand* e altere seu *BandType* para *rbDetail*. Para trabalhar com os resultados no relatório, adicione ao formulário os seguintes componentes realizando as devidas configurações a seguir:

- *SQLDataSet* ("sqlGroup") da paleta *dbExpress*, utilize a *Unit* do *Data Module* e ligue sua propriedade *SQLConnection* ao *sqlConexao* previamente configurado. Adicione em *CommandText* a instrução de *SQL da* **Listagem 3**;
- *DataSetProvider* ("dspGroup") da paleta *Data Access* e ligue-o através da propriedade *DataSet* ao *sqlGroup*;
- *ClientDataSet* ("cdsGroup") da paleta *Data Access* configurando seu *ProviderName* para *dspGroup*. Com um duplo clique sobre o componente abra o *Fields Editor* e adicione todos os campos resultantes da pesquisa *SQL*.

Arraste agora para dentro do componente *QRGroup* um componente *QRDBText* e ligue sua propriedade *DataSet* ao *cdsGroup*

e *DataField* para DEPARTMENT. Selecione o componente *cdsGroup* para que possamos fazer uma configuração indispensável antes de continuar com a criação do relatório agrupado. Precisamos definir o campo pelo qual iremos criar um índice para formação dos grupos, onde na propriedade *IndexFieldNames* você deverá digitar o valor DEPARTMENT, que será o campo que utilizaremos para a formação dos grupos.

Retornando ao formulário de criação do relatório, arraste agora para dentro do *QRBand* mais três componentes *QRDBText*, relacionando a propriedade *DataSet* de ambos ao *cdsGroup* e o *DataField* respectivamente para *FULL_NAME*, *PHONE_EXT* e *SALARY*. Selecione novamente o *QRGroup* e em sua propriedade *Expression* digite o campo pelo qual nosso relatório será agrupado, informando-lhe o seguinte valor:

```
ClientDataSet1.DEPARTMENT
```

Antes do *Preview* selecione ainda o objeto *QuickReport1* de modo que sua propriedade *DataSet* seja configurada ao *cdsGroup*. Realizadas todas as configurações acima descritas, utilize o botão direito do mouse sobre o relatório e selecione a opção *Preview* que deverá originar um relatório semelhante ao exibido na **Figura 7.**

Retornando ao formulário principal para a codificação do *Button* "Agrupado", adicione ao evento *OnClick* do mesmo o seguinte bloco de códigos:

```
Application.CreateForm(TQuickReport1,QuickReport1);
QuickReport1.cdsGroup.Open;
QuickReport1.Preview;
```

> **Nota**: Acesse o menu *Project|Options* e na guia *Forms* remova o objeto *Quick Report1* da seção *Auto Create Forms* enviando-o para *Available Forms*.

Conclusão

Durante este artigo vimos algumas dicas interessantes de como criar rapidamente um relatório utilizando componentes de acesso a dados e QuickReport de forma simples e também com a aplicação de filtros ao relatório.

Criamos também um relatório com a formação de grupos de registros, que facilita muito na análise das informações de um relatório. Dentre estas opções, descubra várias outras funcionalidades do QuickReport, como personalização de campos em *runtime* ou até mesmo a criação de um relatório *master/detail*. Grande abraço e até a próxima. ●

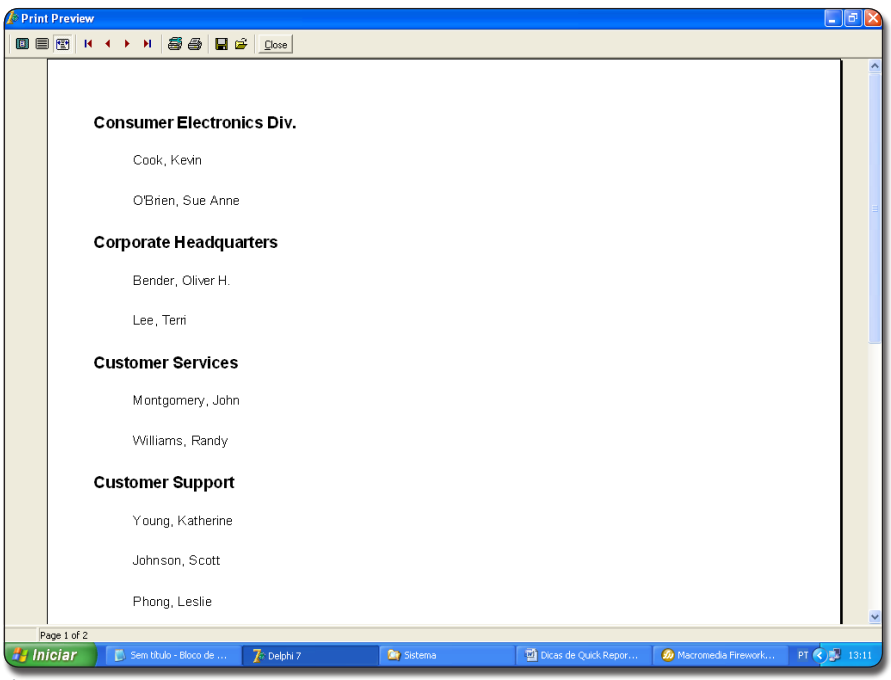
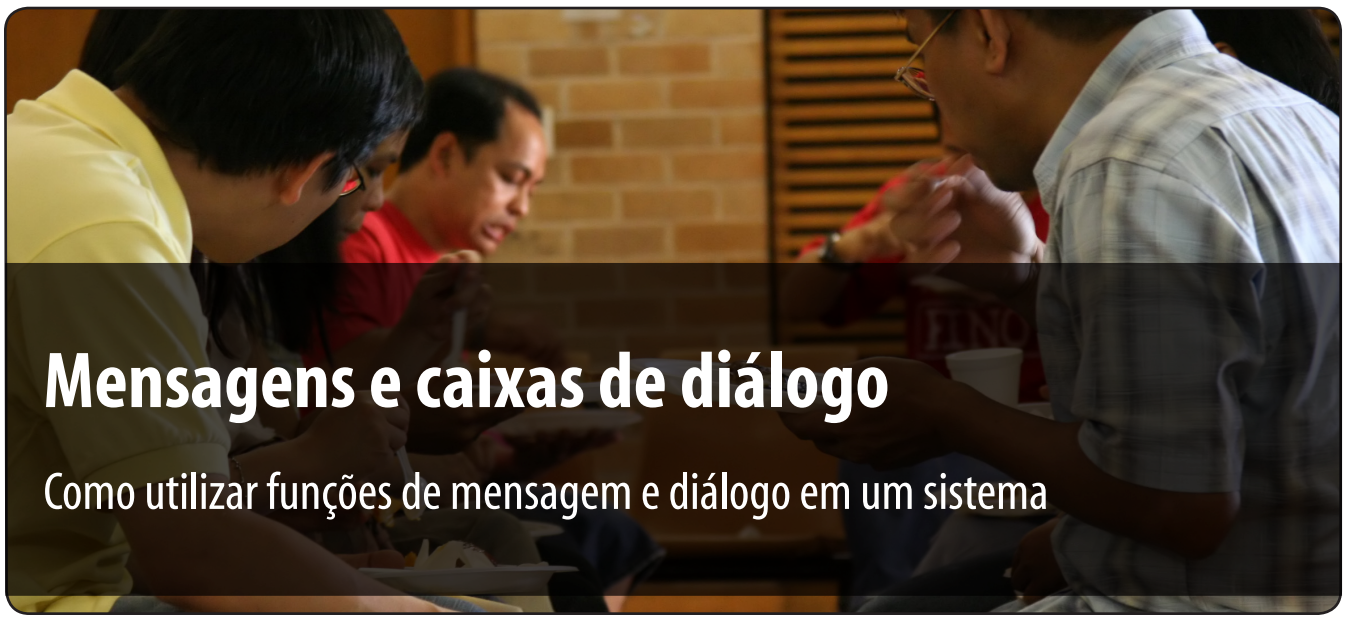


Figura 7. Preview do relatório agrupado por departamentos

Nesta seção você encontra artigos para iniciantes na linguagem Delphi



Mensagens e caixas de diálogo

Como utilizar funções de mensagem e diálogo em um sistema

Adriano Santos
(falecom@adrianosantos.pro.br)
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Ao longo dos meus quase 10 anos de experiência na área de desenvolvimento, adquiri muito conhecimento e tive a oportunidade de tirar minhas próprias conclusões a respeito da programação de sistemas como um todo. Hoje, penso que um software bem desenvolvido precisa seguir alguns critérios e responder a alguns requisitos básicos pra se encaixar bem com as necessidades do cliente e, especialmente, de seus usuários. Um programa não precisa ser complexo demais, nem fazer uso, obrigatoriamente, das últimas tecnologias e tendências de mercado. Na minha opinião acredito que um bom software deve:

- Ser estruturalmente bem programado, no que diz respeito a banco de dados;
- Ter um bom e contínuo planejamento para que as regras de negócios estejam sempre a contento do objetivo principal do sistema;
- Ser informativo, ou seja, deve prover o máximo de informação possível para o usuário final.

Esse último requisito, a meu ver, é um dos mais importantes, pois torna o software atrativo, fácil de usar e inteligente. O que quero dizer é que as mensagens em um sistema são importantíssimas para o bom funcionamento do mesmo. Mensagens de alerta, avisos, perguntas, erros, tudo ou quase tudo deve ser explanado para usuário final.

Nesse artigo veremos os principais tipos de mensagens, como e onde usá-las, as diferenças entre cada uma, caixas de diálogo e criação de exceções em tratamento de erros.

Montaremos um *super-exemplo* para entendermos na prática o uso de *ShowMessage*, *MessageDlg*, *MessageBox*, *OpenDialog*, *SaveDialog* etc. Veremos também, rapidamente, como tratar erros e criar exceções.

Entendendo e usando ShowMessage, MessageDlg e MessageBox

Para começar vejamos primeiramente como funcionam estas três funções usadas para exibir mensagens ao usuário começando por *ShowMessage*.

ShowMessage é uma função que exibe uma mensagem em tela, normalmente utilizada apenas para exibir alguma mensagem simples que não exige ícone ou pergunta. Por isso seu único parâmetro é o texto que será exibido ao usuário final. A seguir podemos ver sua declaração original prevista na *Unit Dialogs* do Delphi.

```
procedure ShowMessage(const Msg: string);
```

Seu uso também é simplificado. Para testarmos a função, bem como as demais que veremos, entre no Delphi e crie uma nova aplicação usando o menu *File|New>Application* ou utilize o projeto que o Delphi cria automaticamente pra nós ao abrir o *IDE*. Salve o formulário como "Mensagens.pas" e a aplicação como "Msgs.dpr".

Insira um componente *GroupBox* da paleta *Standard* e aumente sua altura e largura para que possamos colocar botões grandes dentro dele. Após isso insira um *Button*("btnShowMessage") também da paleta *Standard* dentro do *GroupBox* e modifique seu *Caption* para "ShowMessage". Aproveite e insira outros dois *buttons* no *GroupBox* mudando seus nomes para "btnMessageDlg" e "btnMessageBox". Seus *Captions* deverão ficar como seus nomes retirando apenas o prefixo *btn* do início do nome. Se preferir veja a sugestão de layout na **Figura 1**.

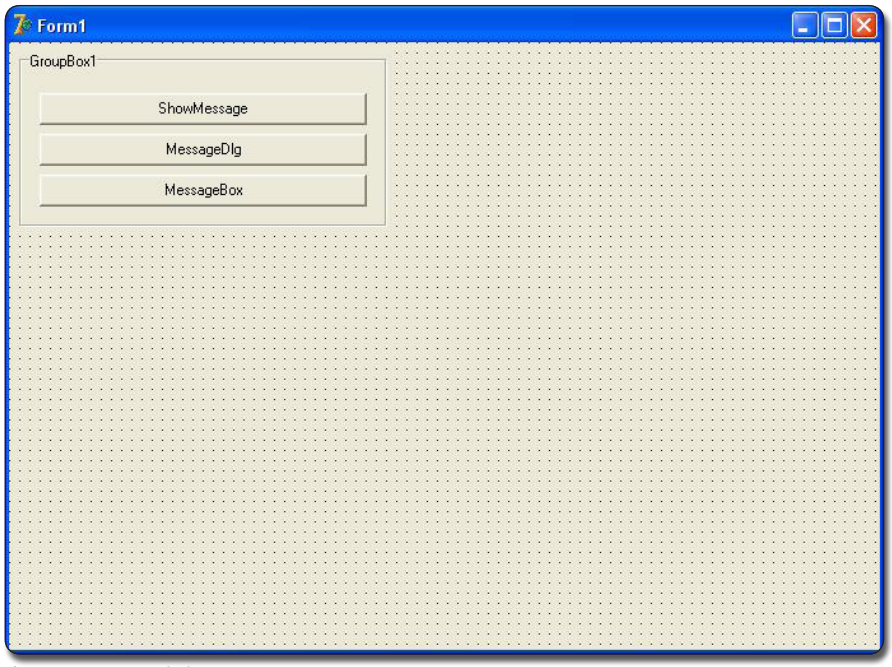


Figura 1. Sugestão de layout

Clique duas vezes no *btnShowMessage* e digite o código a seguir em seu evento *OnClick*.

```
ShowMessage('Exemplo simples de mensagem com
"ShowMessage".');
```

Note que o uso é bastante simplificado como já mencionado, bastando apenas passar para à função qual o texto será exibido. Nesse caso a frase "Exemplo simples de mensagem com "Show Message".". O resultado podemos ver na **Figura 2**.

Como podemos notar, a função *Show Message* exibe um texto simples com apenas o botão *Ok* no centro da mensagem. Uma coisa interessante a se observar é que o título da janela exibe a palavra *Msgs*, exatamente o nome que colocamos para o projeto. O *ShowMessage* captura o nome da aplicação e o coloca automaticamente no título da mensagem.

Figura 2. ShowMessage sendo executada

Pois bem, podemos usar o *ShowMessage* em qualquer ocasião, mas se precisarmos exibir um ícone de atenção ou alerta precisamos recorrer a algo mais sofisticado.

Vamos à função *MessageDlg* que também é declarada na *Unit Dialogs* do Delphi. Sua assinatura padrão pode ser vista logo a seguir:

```
function MessageDlg(const Msg: string;
  DlgType: TMsgDlgType;
  Buttons: TMsgDlgButtons; HelpCtx:
  Longint): Integer;
```

> **Nota**: Assinatura é a declaração original de uma *procedure* ou *function*, ou seja, seu nome e parâmetros de entrada bem como possíveis retornos.

Podemos perceber de início que já complicou um pouco mais. Porém nem tanto assim, vejamos o que cada parâmetro significa, assim temos uma visão macro sobre suas funcionalidades. Seus parâmetros são:

- *Msg* : Esse parâmetro é do tipo *string*, ou seja, texto. Contém a mensagem que será exibida ao usuário do sistema;
- *DlgType*: Tipo de mensagem, também especifica qual ícone aparecerá na tela; A declaração desse tipo diz que podemos declarar como:
  - *mtWarning*: Aviso;
  - *mtError*: Erro;
  - *mtInformation*: Informação;
  - *mtConfirmation*: Confirmação;
  - *mtCustom*: Personalizado;
- *Buttons*: Botões da mensagem. Podem ser:
  - *mbYes*: Sim;
  - *mbNo*: Não;
  - *mbOK*: Ok;
  - *mbCancel*: Cancelar;
  - *mbAbort*: Abortar;
  - *mbRetry*: Repetir;
  - *mbIgnore*: Ignorar;
  - *mbAll*, *mbNoToAll* e *mbYesToAll*: "Todos", "Não para todos" e "Sim para todos";
  - *mbHelp*: Ajuda;
- *HelpCtx*: Contexto de *Help*, ou seja, o valor numérico de atalho para o *help* do sistema se existir um;

Vejamos na prática. Clique duas vezes no segundo botão e digite o código a seguir:

```
MessageDlg('Mensagem simples usando
"MessageDlg".', mtInformation, [mbOk, mbNo], 0);
```

Nesse caso estamos definindo que uma mensagem com a frase "Mensagem simples usando "MessageDlg"." será exibida na tela. O ícone usado é o de informação ("mtInformation") e o botão é o *Ok* ("mbOk"). Como não temos um *help*, passamos como *0* o último parâmetro.

Antes de visualizarmos o resultado vamos a explicação e uso da terceira função, *MessageBox*. *MessageBox*, visualmente, tem a mesma aparência que a *MessageDlg*, com uma leve diferença que já vamos notar de início. Inicialmente, quais são e como usar seus parâmetros?

• *hWnd*: Parâmetro do tipo *HWND*. Esse parâmetro é o *Handle* da janela. Podemos passar de diversas formas, tais como: *Handle*, *Application.Handle* ou *0* ("zero");

• *lpText*: Do tipo *PAnsiChar*. É um *string* também, porém terminado com o caracter nulo (*#0*). Quando precisamos passar mais de uma frase para ele é necessário envolvermos as frases dentro de uma função de conversão, *PChar*. Veremos mais a seguir;

• *lpCaption*: É o título da mensagem. Nesse caso devemos dizer à função qual a palavra/frase que deverá ser colocada no título;

• *uType*: *Cardinal*. É aqui que escolhemos qual o tipo de mensagem (ícone), botões e *defaults*.

Figura 3. MessageDlg e MessageBox

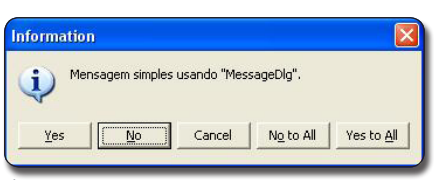


Figura 4. MessageDlg com vários botões

> **Nota**: Toda janela e/ou objeto no Windows recebe um número de *Handle* ao ser instanciado ("criado"). É basicamente seu endereço de memória para que o Sistema Operacional possa fazer referência a ele.

Para entendermos melhor digite o código em seguida no evento *OnClick* do terceiro botão. Logo após execute a aplicação e clique nos botões *MessageDlg* e *MessageBox*. Veja as diferenças na **Figura 3**.

```
MessageBox(Handle, 'Mensagem simples usando
  "MessageBox"', 'Texto do título', MB_YESNO +
   MB_ICONINFORMATION);
```

A diferença mais nítida que podemos ver está justamente nos botões de ambas as mensagens. A função *MessageDlg* é nativa do Delphi e exibe sempre os botões em inglês ao contrário da função *MessageBox*, declarada na *Unit Windows*, que se adapta a linguagem do sistema operacional onde está rodando, ou seja, como sistema operacional onde tirei as fotos para este artigo estava em português automaticamente os botões são exibidos como "Sim" e "Não". Se testássemos em uma cópia em inglês do Windows, esses botões estariam como no *MessageDlg*.

A segunda diferença é mais sutil, mesmo assim podemos notar. O título de ambas é diferente. Como em *MessageDlg* não definimos qual o título será exibido, o sistema pega pelo tipo de mensagem, nesse caso de informação (*Information*). Já na *MessageBox* definimos no segundo parâmetro que o título deveria conter a mensagem "Texto do título".

Em se tratando de codificação temos diferenças gritantes, pois os parâmetros são bastante incomuns. Um exemplo disso são os parâmetros de tipo de mensagem e botões escolhidos para exibição. Em *MessageDlg* definimos o tipo da mensagem e os botões separadamente.

No segundo parâmetro informamos o tipo de mensagem, *mtInformation*, e no parâmetro seguinte os botões que aparecerão. Nesse caso passamos "Sim" e "Não" separados por vírgula e envolvidos em colchetes, ex: *[mbYes, mbNo]*. Se quiséssemos mais botões, bastaria incluir novos itens, como no caso a seguir onde mostramos os botões "Sim", "Não", "Não para todos", "Sim para todos" e "Cancelar". O resultado seria como na **Figura 4**.

```
MessageDlg('Mensagem simples usando
 "MessageDlg".', mtInformation, [mbOk,
  mbYes, mbNoToAll, mbYesToAll, mbCancel], 0);
```

Em *MessageBox* não é possível obter o mesmo resultado. Porém outras vantagens são obtidas ao usar essa função. Caso seja necessário definir um botão diferente como *default* ("padrão") isso só será possível com *MessageBox*. Imagine uma mensagem de confirmação: "Deseja excluir o arquivo atual?". E nessa mensagem você queira definir que o segundo botão será o foco principal da mensagem, evitando que o usuário cometa erros. Como faríamos?

Pois bem, com *MessageBox* é possível escolhermos o botão *default*, veja:

```
MessageBox(Handle, 'Confirma exclusão do
  arquivo', 'Texto do título', MB_YESNO +
  MB_ICONINFORMATION + MB_DEFBUTTON2);
```

Observe que incluímos uma nova constante: *MB_DEFBUTTON2*. São quatro ao todo. Nesse caso estamos dizendo que o botão que aparecerá em foco no momento que a mensagem abrir será o segundo botão, o *Não*.

Se quisermos incluir mais de uma linha em uma mensagem basta usarmos o caractere *13* que equivale ao *Enter* do teclado. Veja:

```
MessageBox(Handle, PChar('A exclusão
  deste arquivo será permanente.' + #13 +
  'Confirma a exclusão do arquivo'), 'Texto
   do título', MB_YESNO + MB_ICONINFORMATION +
   MB_DEFBUTTON2);
```

E já que estamos falando de erros de usuário, vejamos rapidamente como fazer uso das mensagens para desviar o fluxo do sistema e tratar alguns erros.

Tratando erros e desviando o fluxo

Certamente um sistema sem tratamento de erros e/ou desvio de fluxo de tarefas se torna um tanto quanto perigoso e não muito inteligente. O que veremos a seguir é como fazer uso das mensagens antes explicadas para prever as ações

do usuário e desviá-lo para outra tarefa, bem como a criação de exceções.

Insira um novo *GroupBox* ao formulário e nele um novo *Button*("btnMessage DlgComIf") com o *Caption* igual a "MessageDlg com If". Logo abaixo coloque um novo *Button*("btnMessageBoxComIf") e troque seu *Caption* para "MessageBox com If". Por fim adicione um *Label*("lblResultadoMsg") abaixo deste último botão. A seguir veremos a assinatura das três funções que vimos na seção anterior:

```
procedure ShowMessage(const Msg: string);
function MessageDlg(const Msg: string;
  DlgType: TMsgDlgType;
  Buttons: TMsgDlgButtons; HelpCtx:
  Longint): Integer;
function MessageBox(hWnd: HWND; lpText:
  PAnsiChar; lpCaption: PAnsiChar;
  uType: Cardinal): Integer;
```

Perceba que na verdade *ShowMessage* não é uma *function* ("função") e sim uma *procedure* ("procedimento") o que significa que não retorna resultado, diferentemente das demais funções que retornam um valor do tipo *Integer*, ou seja, inteiro. É aí que está a parte interessante de tudo. Com *MessageDlg* ou *MessageBox* podemos "perguntar" ao usuário o que ele deseja fazer. Pegamos a sua resposta e desviamos o fluxo do sistema para outra parte dele. Experimente, digite o código abaixo no evento *OnClick* do *btnMessageDlgComIf*.

```
if MessageDlg('Você deseja exluir este
  arquivo?', mtConfirmation,
  [mbYes, mbNO], 0) = mrYes then
  lblResultadoMsg.Caption :=
  'Sim, o usuário deseja excluir o arquivo'
else
  lblResultadoMsg.Caption :=
   'Não, o usuário desistiu';
```

Salve, execute a aplicação e clique no botão "MessageDlg com If". Fizemos uma pequena verificação. Se o usuário escolher "Sim", o resultado da função é igual a *mrYes*, ou seja, o *Label* é atualizado com a mensagem "Sim, o usuário deseja excluir o arquivo", caso contrário a mensagem é "Não, o usuário desistiu.". A constante *mrYes* está declarada na *Unit Control* do Delphi e contém entre vários retornos, as opções *mrNo* ("Não") e *mrOk* ("Ok").

Também é possível obtermos o mesmo resultado utilizando *MessageBox*, porém os retornos são diferentes, veja:

```
if MessageBox(Handle, 'Você deseja excluir
  este arquivo?', 'Exclusão', MB_YESNO +
  MB_ICONQUESTION) = ID_YES then
   lblResultadoMsg.Caption :=
   'Sim, o usuário deseja excluir o arquivo'
else
  lblResultadoMsg.Caption :=
  'Não, o usuário desistiu';
```

O esquema é o mesmo, verificamos se o usuário clicou em *ID_YES* equivalente ao "Sim" e então atualizamos o *Label* conforme necessário. (**Figura 5**).

Agora vamos ver como aplicar mensagens em tratamentos de erros. Arraste para o formulário mais um *GroupBox* e dentro dele crie um *layout* como mostrado na **Figura 6**.

Dê o nome de "btnValidar" ao botão e "lblConversao" ao *Label* e "edtNumero" ao *Edit*, o restante mantenha os nomes padrão. No evento *OnClick* do botão digite o código da **Listagem 1**.

Utilizamos nesse código um bloco *try.. except* para validar os dados. Estamos tentando fazer uma conversão de *string* (texto) para *Integer* (número) usando a função *IntToStr*. Caso a conversão seja efetuada, o *lblConversao* é atualizado e recebe o *Caption* igual a "Convertido com sucesso". Caso contrário o *MessageDlg* é executado e uma mensagem mostrada.

Uma outra abordagem nesse mesmo exemplo seria fazer com que o próprio sistema criasse a exceção gerando a mensagem automaticamente. Modifique o código do evento de forma que fique como na **Listagem 2**.

Perceba que trocamos a *MessageDlg* pela linha:

```
raise Exception.Create('Erro de conversão');
```

Isso fará com que a aplicação gere uma exceção e conseqüentemente uma mensagem de erro ao usuário final. Normalmente utilizamos essa abordagem em sistemas orietados a objetos, já que dentro das classes não podemos conversar diretamente com a camada externa.

Caixas de diálogo

Nessa última parte do artigo veremos como funcionam os componentes da paleta *Dialogs*. Esses componentes são usados quando é necessário abrir e salvar arquivos, capturar imagens do disco,

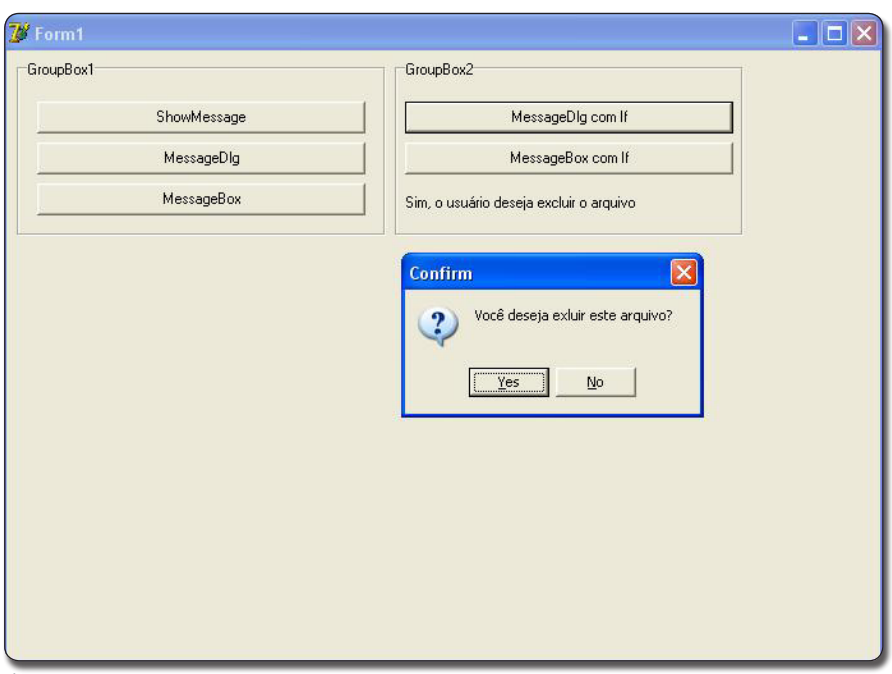


Figura 5. Exemplo de condicional com MessageDlg

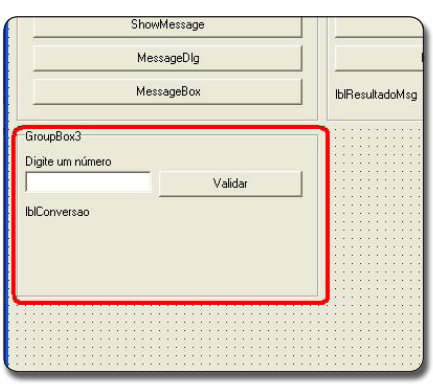


Figura 6. Sugestão de layout

configurar a impressora etc.

Para este exemplo criaremos um mini-editor de textos onde será possível abrir, salvar, configurar a impressora, fonte e cor de fundo do texto. Ainda poderemos selecionar uma foto em formato JPEG para exibir à direita da tela.

Clique em *File|New>Application* e crie uma nova aplicação no Delphi salvando-a como "Dialogos.dpr". O formulário principal salve-o como "uPrincipal.pas" e troque seu *Name* para "frmPrincipal". Desenhe uma tela semelhante à **Figura 7**.

Na parte superior temos os botões "btnAbrir", "btnSalvar", "btnConfigurar", "btnFonte" e "btnImagem", todos com seus respectivos *Captions*.

Na parte central do formulário insira um *Memo*("memTexto") da paleta *Standard* e um *Image*("imgImagem") da paleta *Additional*. Por fim adicione um novo componente *Button*("btnCorFundo") logo abaixo do *memTexto*.

Perceba que inseri também seis componentes da paleta *Dialogs*. Cada um deles terá uma função crucial em nosso exemplo. Tive o cuidado de manter os nomes originais dos componentes, desta forma fica mais fácil identificar qual tipo de componente devemos selecionar na paleta. A seguir veja uma breve explicação de cada um:

- *OpenDialog1*: Este componente é responsável por abrir arquivos. Sua propriedade *Filter* prevê quais tipos de arquivos podem ser abertos;
- *SaveDialog1*: Ao contrário do *OpenDialog*, o *SaveDialog* nos permite salvar arquivos no disco. Ele também possui a propriedade *Filter* para a mesma finalidade;
- *FontDialog1*: Esse componente, quando executado, permite modificar a propriedade fonte de determinados componentes de acordo com a fonte selecionada em sua tela;
- *PrinterSetupDialog1*: Como o próprio nome sugere, esse componente pode ser usado para configurar a impressora;
- *ColorDialog1*: É possível modificar a propriedade *Color* de objetos usando a propriedade de mesmo nome deste componente;
- *OpenPictureDialog1*: Semelhante ao *OpenDialog*, porém esse componente permite visualizar a foto antes de selecioná-la;

Vamos codificar o exemplo e entender como funcionam esses componentes começando por *OpenDialog1* e *SaveDialog1*.

Ambos componentes possuem as propriedades:

- *DefaultExt*: É a extensão *default* ("padrão"). Pode ser usada em conjunto com a propriedade *Filter* para filtrar os tipos de arquivos que aparecerão na caixa de diálogo;
- *Filter*: É o filtro de arquivos que podem ser abertos no sistema. Podemos especificar quais tipos de arquivos serão compatíveis com nosso sistema;
- *InitialDir*: Diretório inicial da caixa de diálogo. Especificando esta propriedade com uma pasta válida, o sistema fará a seleção desta pasta automaticamente;

Vamos entender melhor tudo isso. Vamos atribuir valores às propriedades *Filter* e *InitialDir* do componente *OpenDialog*. Clique duas vezes na propriedade *Filter* e note que uma caixa de diálogo é aberta **(Figura 8).**

Observe que na **Figura 8** já existem

Listagem 1. Evento OnClick do btnValidar

```
procedure TForm1.btnValidarClick(Sender: TObject);
begin
  try
    StrToInt(edtNumero.Text);
    lblConversao.Caption := 'Convertido com sucesso';
  except
    MessageDlg('Erro de conversão', mtError,
      [mbOk], 0);
  end;
end;
```

Listagem 2. Evento OnClick do btnValidar modificado para gerar exceção

```
procedure TForm1.btnValidarClick(Sender: TObject);
begin
  try
    StrToInt(edtNumero.Text);
    lblConversao.Caption := 'Convertido com sucesso';
  except
    raise Exception.Create('Erro de conversão');
  end;
end;
```

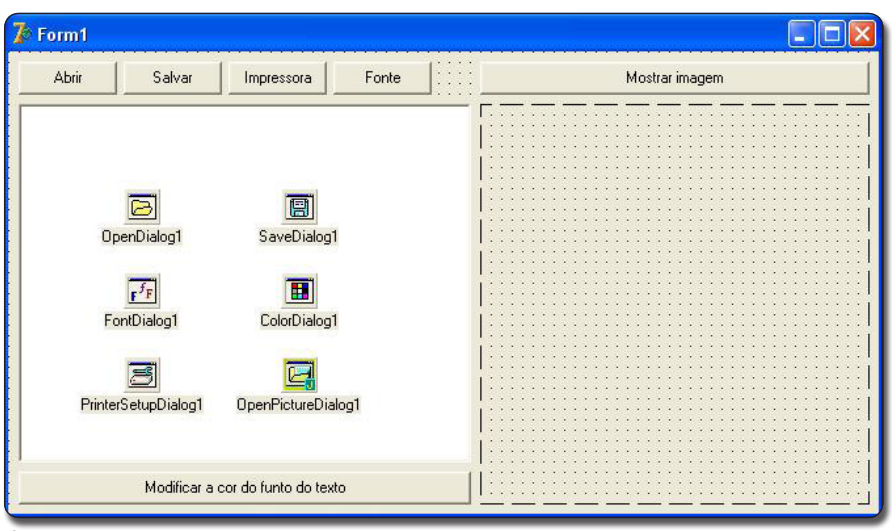


Figura 7. Exemplo de tela para o mini-editor

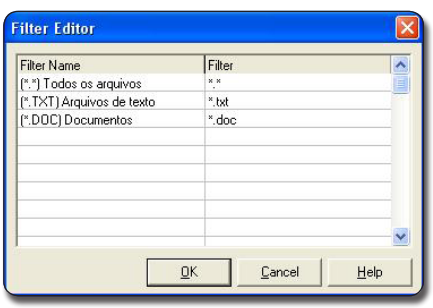


Figura 8. Edição da propriedade Filter

valores digitados, pois incluí alguns itens, porém o padrão desta caixa é vir sempre sem itens. A primeira vez que a propriedade é aberta a grade está vazia para que possamos configurar como desejamos. Pois bem, a coluna da esquerda, *Filter Name*, é a frase que aparecerá para o usuário final. Já a coluna da direita é o filtro propriamente dito. Portanto a sua esquerda digite "(*.*) Todos os tipos de arquivos" e a esquerda insira o filtro "*.*". Nas linhas posteriores insira novos valores conforme o que vimos na **Figura 8** e então confirme.

Para a propriedade *InitialDir* crie uma pasta no C:\ e atribua seu nome a esta propriedade. No meu caso coloquei como *C:\Temp*. O importante é termos essa pasta criada no Windows para que o componente possa selecioná-la.

Repita esses passos para o componente *SaveDialog* e vamos à codificação.

No evento do botão *btnAbrir* digite o código a seguir e veja como é simples abrir um arquivo via programação.

```
if OpenDialog1.Execute then
  memTexto.Lines.LoadFromFile(
    OpenDialog1.FileName);
```

A explicação é simples. Todos estes componentes possuem o método *Execute* que, evidentemente, executa a caixa de diálogo e mostra-a ao usuário. Mas não é só isso, na verdade quando chamamos este método, a caixa de diálogo é aberta e aguarda a confirmação ou cancelamento do usuário, ou seja, o código que vem logo após o *Then* da nossa condicional só será executado se o usuário final clicar no botão principal da caixa de diálogo. Nos casos do *OpenDialog* e *SaveDialog* são "Abrir" e "Salvar", respectivamente.

Clicando em *Abrir* chamamos o método *LoadFromFile* da propriedade *Lines* do componente *memTexto*, assim o texto é aberto no *Memo*.

O código do botão "Salvar" é bastante semelhante, veja:

```
if SaveDialog1.Execute then
  memTexto.Lines.SaveToFile(
    SaveDialog1.FileName);
```

Invocamos novamente o método *Execute* do componente de diálogo e caso seja confirmado chamamos o método *SaveToFile*, também da propriedade *Lines* de nosso *Memo*.

Como podemos ver, não há segredos na abertura e gravação de arquivos. Experimente compilar o projeto e executá-lo. Clique nos botões que codificamos e veja o resultado. Na **Figura 9** podemos ver os tipos de arquivos que podemos abrir nesse exemplo.

É claro que a caixa de diálogo não faz nada sozinha. A indicação no filtro "(*.DOC) Documentos" não significa sucesso na abertura deste arquivo. Nesse caso daria erro na chamada do método *LoadFromFile* ou *SaveToFile*, já que o componente *Memo* não é compatível com documentos do Word.

Por isso é necessário que nossa aplicação seja desenvolvida de maneira que se compatibilize com os tipos de arquivos

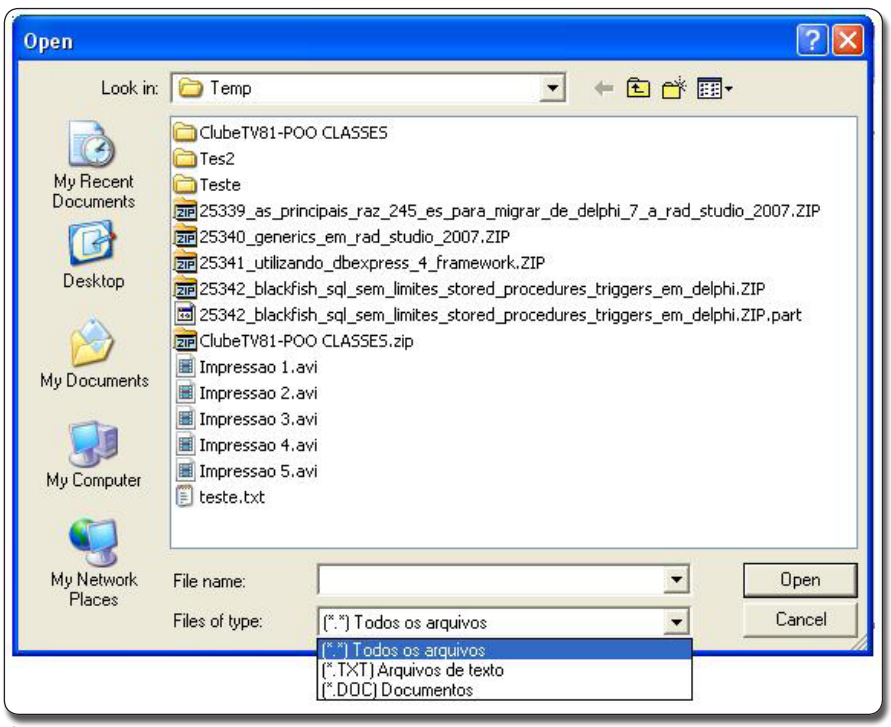


Figura 9. Caixa de diálogo OpenDialog

que desejamos abrir e/ou salvar.

Voltando ao exemplo vamos codificar o botão *Fonte*("btnFonte"). Ao clicarmos nesse botão poderemos configurar o formato de fonte do componente *Memo*. Na verdade atribuiremos os valores da propriedade *Font* do componente à propriedade de mesmo nome do *Memo*. Clique duas vezes no botão mencionado e digite o código da **Listagem 3**.

O que estamos fazendo no código que acabamos de digitar foi atribuir os valores das sub-propriedades *Name*, *Style*, *Color* e *Size* da propriedade *Font* do componente *FontDialog*. Perceba que o componente também possui um método *Execute* e que o mesmo se comporta exatamente como os componentes que já vimos. Salve, compile e execute o programa. Clique no botão *Fonte* e veja o que acontece (**Figura 10).**

Experimente agora digitar o código abaixo no evento *OnClick* do botão "Modificar a cor do fundo do texto".

```
if ColorDialog1.Execute then
  memTexto.Color := ColorDialog1.Color;
```

Igualamos a propriedade *Color* do *Memo* à propriedade de mesmo nome do *ColorDialog1*. O código para configuração de impressoras consiste apenas em chamar o método *Execute* do componente *PrinterSetupDialog1*, veja:

```
PrinterSetupDialog1.Execute;
```

A última coisa que veremos agora é como carregar imagens em um componente *Image* da paleta *Additional*. No evento *OnClick* do botão "Mostrar Imagem" digite o código a serguir:

```
if OpenPictureDialog1.Execute then
  imgFoto.Picture.LoadFromFile(
    OpenPictureDialog1.FileName);
```

Veja que estamos novamente utilizando o método *Execute* juntamente com o *LoadFromFile*. Poderíamos utilizar tranquilamente um *OpenDialog*, porém a principal diferença entre o *OpeDialog* e o *OpenPictureDialog* está no fato de poder visualizar a imagem antes de confirmar a caixa de diálogo.

A única ressalva que faço é que para utilizar-se de imagens no formato JPEG é necessário declarar a *Unit Jpeg* no *Uses* do formulário, do contrário uma exceção será lançada. Veja na **Figura 11** a caixa de diálogo *OpenPictureDialog* em ação.

Listagem 3. Código do botão Fonte

```
procedure TForm1.btnFonteClick(Sender: TObject);
begin
  if FontDialog1.Execute then
  begin
    memTexto.Font.Name := FontDialog1.Font.Name;
    memTexto.Font.Style := FontDialog1.Font.Style;
    memTexto.Font.Color := FontDialog1.Font.Color;
    memTexto.Font.Size :=FontDialog1.Font.Size;
  end;
end;
```

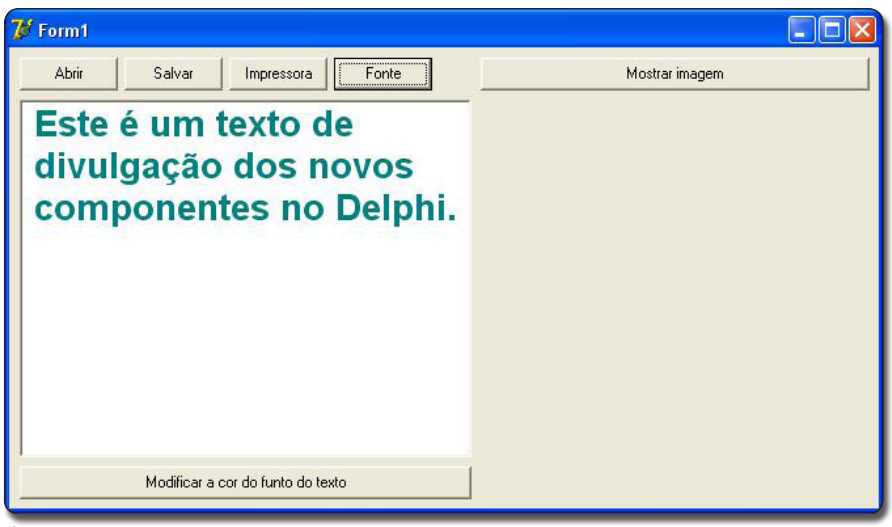


Figura 10. Botão Fonte em execução

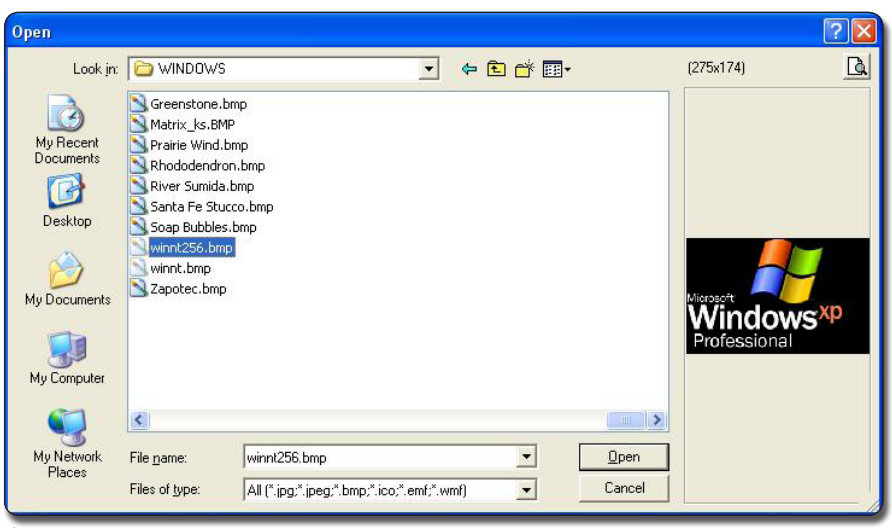


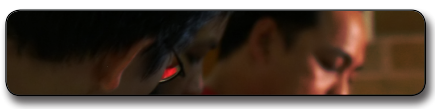
Figura 11. Caixa de diálogo OpenPictureDialog em ação

Conclusão

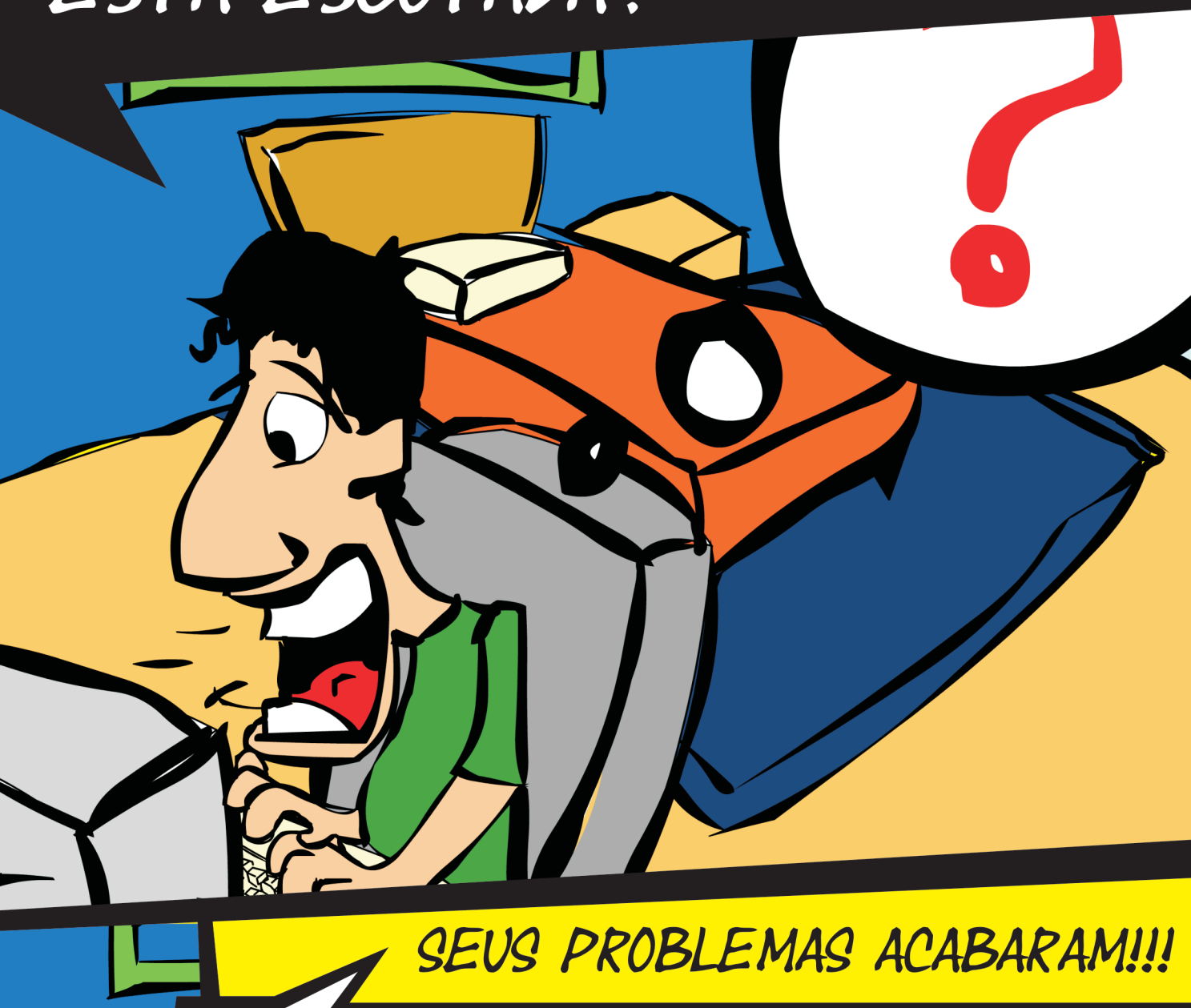
Neste artigo vimos as principais caixas de mensagens e de diálogos utilizadas no dia-a-dia. Seguindo essas orientações é perfeitamente viável tornar o sistema mais comunicativo e informativo.

Existem dezenas de outras formas de se usufruir de todo poder da VCL do Delphi. Procure modificar e estudar outras propriedades das caixas de diálogo. Tente criar suas próprias caixas de mensagem personalizadas.

Lembre-se que o limite é a criatividade. Boa sorte e até a próxima. ●



A EDIÇÃO QUE VOCÊ PRECISA ESTÁ ESGOTADA?



SEUS PROBLEMAS ACABARAM!!!

Seja um assinante Gold!

Com a assinatura Gold você já pode consultar online todos os artigos publicados na sua revista desde a edição nº 1.

DevMedia GROUP

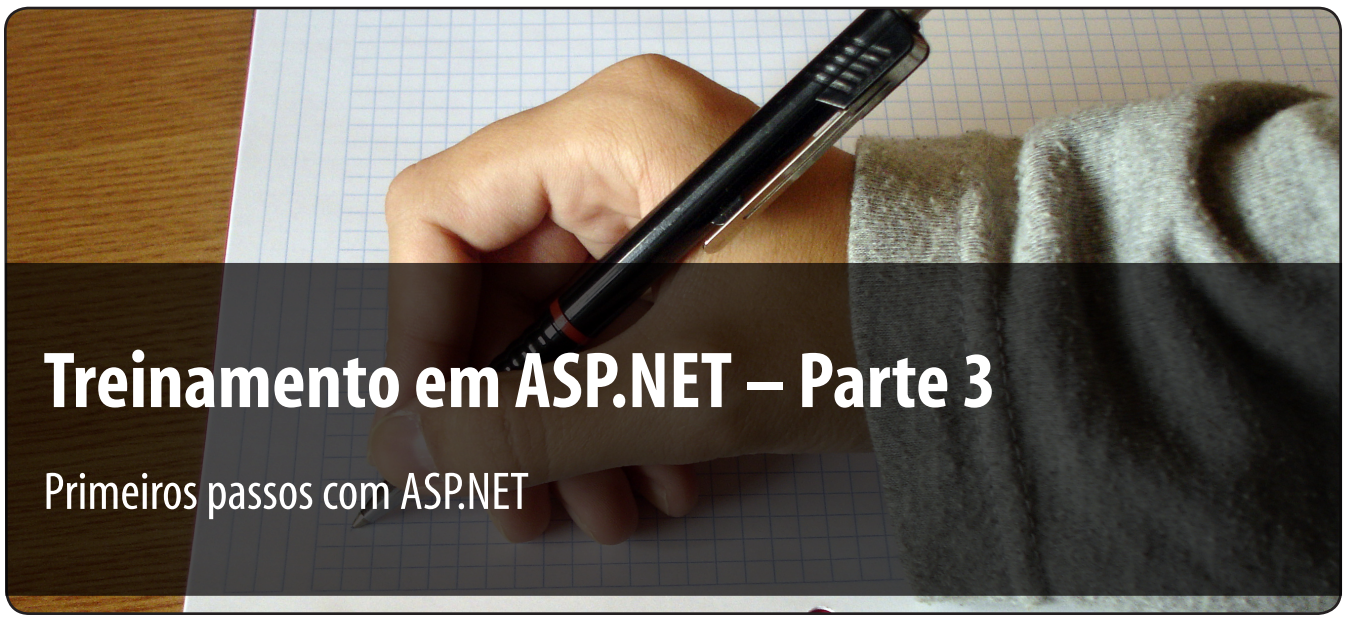
Saiba Mais! Acesse:
www.devmedia.com.br/assgold

Para mais informações:
www.devmedia.com.br/central

Assinatura Gold

Atenção: Já encontram-se disponíveis as assinaturas GOLD das revistas WebMobile e .net Magazine.

Nesta seção você encontra artigos para iniciantes na linguagem Delphi



Treinamento em ASP.NET – Parte 3

Primeiros passos com ASP.NET

Renato Haddad
é Microsoft Most Valuable Professional (MVP). Autor de diversos treinamentos multimídia .NET e SQL Reporting Services para Microsoft Brasil e América Latina.

Esta é a última parte do treinamento em ASP.NET da sessão EasyDelphi. Aqui, você aprenderá o conceito de *Session*, bem como a criar e ler variáveis de sessão, enviar e-mails para um conjunto de clientes selecionados e criar e ler arquivos XML a partir de um *Dataset*. Você verá também como criar arquivos de cabeçalho e rodapé que serão usados em todas as páginas de seu site.

Neste artigo veremos técnicas de conexão com o banco de dados para a montagem de arquivos XML bem como listagem dos clientes para envio de e-mails.

Criando o cabeçalho e rodapé único

Na parte 1 deste treinamento, foi criada uma aplicação chamada *DelphiMag*, que contém todos os arquivos desenvolvidos até aqui. Portanto, abra esta aplicação e adicione um novo *User Control* utilizando-se do menu *File|New|Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET User Control*. Salve-o como "cabecalho.ascx" no diretório da aplicação. Repare na extensão desse arquivo; ele será utilizado como cabeçalho em todas as páginas do site. No *layout* deste documento, adicione uma tabela com 2 linhas ("Rows") e 3 colunas ("Columns") assim como fizemos nos demais artigos do treinamento.

Na primeira célula e na última da primeira linha, arraste um componente do tipo *Image* da paleta *Web Controls* na *Tool Palette*. Configure a propriedade *ImageUrl* de cada *Image* informando o nome da imagem que será utilizada nele. Se preferir clique no botão reticências da propriedade e localize uma imagem no disco.

Nota: As imagens em um site devem ficar em um caminho conhecido das páginas. O ideal é colocá-las sempre na mesma pasta do website ou em um diretório na raiz, tal como: imagens. Nesse caso o caminho da foto/imagem na propriedade *ImageUrl* ficaria algo como *"\imagens\image.gif"*.

Na célula do meio digite um texto de sua preferência. Nesse treinamento usei a frase "Treinamento em ASP.NET". Em seguida insira três componentes do tipo *Hyperlink*, também da paleta *Web Controls*, um em cada célula da segunda linha como podemos ver na **Figura 1**. Modifique a propriedade *ID* de cada *Hyperlink* usando os nomes "hylSessao", "hylGerenciaEmails" e "hylXML", respectivamente. Seguindo da esquerda para a direita modifique a propriedade *NavigateUrl* de cada *Hyperlink* informando "Session.aspx", "GerenciaEmails.aspx" e "ArquivosXML.aspx".

A propriedade *NavigateUrl* é justamente o arquivo (página) que será chamado quando clicado, ou seja, o link em si.

Adicione um novo *User Control* como fizemos anteriormente, mas dessa vez faremos algo mais simples, pois este será o rodapé da página. Salve este *User Control* como "rodape.ascx".

Insira um componente do tipo *HTML Horizontal Rule* da paleta *HMTL Elements*. Ao final dele pressione *Enter* e então digite o texto "Copyright (2007) by ClubeDelphi – Todos os direitos reservados". Com isso, você pode criar apenas um arquivo padrão e usá-lo em qualquer página, eliminando a necessidade de repetir o mesmo código diversas vezes. **Figura 2**

UserControls são formulários Web encapsulados em um controle reutilizável. Eles são usados para armazenar blocos de código repetitivos que muitas páginas de um site *Web* poderão precisar. Em vez de ter que copiar e colar blocos de código repetitivo para fornecer o cabeçalho em todas as páginas, podemos criar um único controle que terá estes blocos de código prontos para serem usados. Eles substituem o modo antigo do ASP de se acoplar funções em outros arquivos utilizando o método *include files*.

Estes controles podem fazer muito mais do que simplesmente gerar cabeçalhos, podem ser atribuídos, por exemplo, a maneira como as páginas irão se comportar diante do usuário. Podem também ser criados atributos e métodos de modo que o controle possa se adaptar dependendo dos valores configurados ou fornecidos pelo usuário ou pelo próprio sistema. Todos os *User Controls* que forem criados devem ter a extensão ".ascx".

Após criados, os *User Controls* ficam disponíveis no *Project Manager*, bastando apenas arrastá-los para o local onde ficarão armazenados na página. Veja um exemplo na **Figura 3** de como ficou nossa tela.

Criando e entendendo o conceito de Session

O conceito de sessões é fundamental para que você usufrua os recursos que o ASP.NET e a internet proporcionam às aplicações. Sempre que você abre um site, é criada uma sessão no servidor para cada internauta. Isso significa que, se dez mil usuários acessarem o mesmo site ao mesmo tempo, serão criadas dez mil *sessões*. Portanto, não há nenhum problema em manter diversas *Sessions*, mesmo porque isso já é feito de forma automática no servidor.

A sessões em diversas linguagens de programação, incluindo o ASP.NET, são chamadas de *Session*. Como uma

Figura 1. Arquivo de cabeçalho

Figura 2. Arquivo de rodapé

Figura 3. Exemplo de página com User Control

Session não enxerga a outra, é possível trabalhar com os dados de seus respectivos internautas. Todas as variáveis de sessão criadas poderão ser enxergadas em qualquer página até que o internauta feche o site ou que a conexão caia. Em resumo, uma *Session* é uma área restrita para cada usuário. Veja por exemplo como adicionar uma sessão:

```
Session.Add('Nome', 'Adriano Santos');
```

Vamos criar uma nova página e entender melhor, na prática, como é usar sessões. Crie um novo formulário usando a opção *File|New|Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Pages* e chame-o de "Session.aspx" salvando-o na página de nosso projeto *DelphiMag*.

Para inserir o cabeçalho que criamos anteriormente apenas arraste o *User Control cabecalho.ascx* para a página recém criada e repita esse procedimento para o arquivo "rodape.ascx". Desenhe uma tela semelhante à **Figura 4**. Foram inseridos dois componentes *TextBox* da paleta *Web Controls* com os *ID's* igual a "edtNome" e "edtSenha", respectivamente. Para que o campo *Senha* fique com os caracteres ocultos ao digitar o texto modifique a propriedade *TextMode* para *Password*.

Logo em seguida três *Buttons*, também da paleta *Web Controls*, foram adicionados. Seus nomes (ID's) são: "btnCriar", "btnLer" e "btnAbandona". Incluímos ainda um *Label*("lblSessao") e um Hyperlink("hylGerenciaEmails"). Além da propriedade *ID*, do *hylGerenciaEmails*, é necessário configurar a propriedade *NavigateUrl* (página para onde o link aponta) com o valor "GerenciaEmails" e seu *Text* para "Gerencia E-mails". Veja na **Figura 4** o layout do formulário em tempo de design.

> **Nota**: Insira os componentes entre o primeiro e o segundo User Control, ou insira o "cabecalho", os componentes e, por último, o rodapé.

Listagem 1. Códigos dos botões

```
procedure TWebForm1.btnCriar_Click(sender: System.Object; e: System.EventArgs);
begin
  { Verifica se existem dados digitados }
  if (edtNome.Text.Trim <> '') or
    (edtSenha.Text.Trim <> '') then
  begin
    { Cria variáveis de Session }
    Session.Add('Usuario', edtNome.Text.Trim);
    Session.Add('Senha', edtSenha.Text.Trim);
    lblSessao.Text := 'Variáveis criadas na Sessão';
  end
  else
    lblSessao.Text := 'A Sessão não foi criada';
end;
```

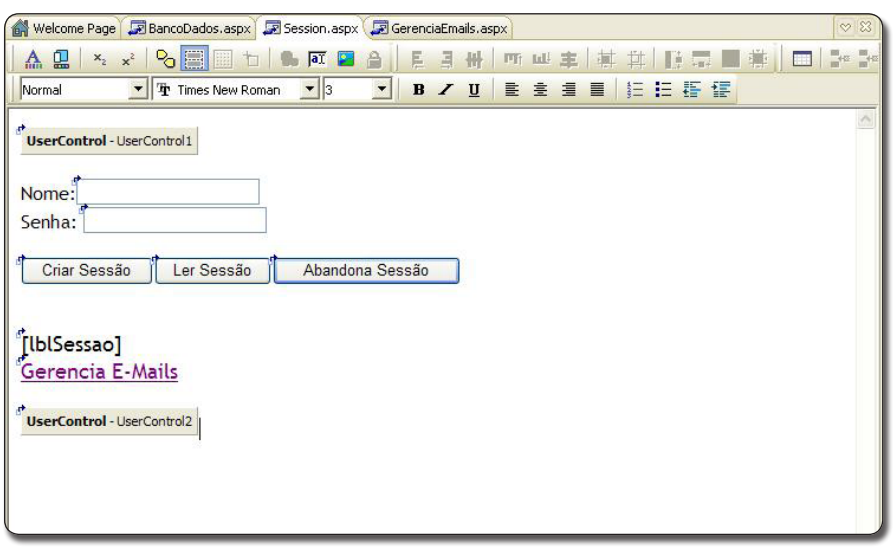


Figura 4. Layout final do Session.aspx

Dê um duplo clique em *btnCriar* e digite o código da **Listagem 1**. O código é bem simples de entender; apenas fazemos uma checagem com *IF* e verificamos se o campo *Nome* ou *Senha* possuem valor. Em caso negativo enviamos uma mensagem ao usuário, caso contrário criamos duas variáveis de sessão usando o método *Add* do objeto *Session*. No método informamos o nome da sessão no primeiro parâmetro e no segundo o valor dele, que neste caso é o *nome* ou *senha* dígitos nos seus respectivos campos. Por último atualizamos o *lblSessao* com a frase "Variáveis criadas na Sessão".

Agora nos demais botões a coisa fica ainda mais simples. Vejamos como resgatar os valores gravados na sessão e como apagá-los. No *btnLer* digite o código em seguida:

```
blSessao.Text := 'ID da Sessão: ' +
  Session.SessionID + ' - Usuário: ' +
Session.Item['Usuario'].ToString +
  ' - Senha: ' + Session.Item['Senha'].ToString;
```

O código que vimos imprime o valor dos campos *Usuário* e *Senha* gravados anteriormente através do botão *btnCriar*. Estamos usando também a propriedade *SessionID* do objeto *Session* para retornar o *ID* da sessão. Esse ID é único, ou seja, cada usuário tem o seu. Para finalizarmos a sessão basta chamarmos o método *Abandon*, também do objeto *Session*, e limpar o *Text* do *Label* como podemos ver a seguir:

```
Session.Abandon;
lblSessao.Text := '';
```

Defina o formulário *Session.aspx* como *default* usando a opção *Set As Start Page* ao clicar com o botão direito do mouse no arquivo. Salve o projeto, compile-o e o execute-o para ver o resultado. A **Figura 5** exibe o formulário no navegador com o cabeçalho, o rodapé e todos os controles do *Session.aspx*.

Digite o nome, a senha e clique nos botões correspondentes para testar. O objetivo de criarmos uma *Session* é que, nas próximas páginas a serem criadas (veja mais adiante), seja verificado se a *Session* existe. Essa será a forma de validar o internauta a navegar somente em páginas autorizadas. Você notará nestas outras páginas carregadas que esse teste

já foi executado no evento *Page_Load*. Se não existir a variável criada para sessão *Usuário*, o internauta será redirecionado à página *Session.aspx* para efetuar o *login*.

Criando o banco de dados

Para entendermos melhor e testarmos a próxima parte, *Envio de E-mails*, desse artigo vamos criar um banco de dados e nele uma tabela contendo clientes e e-mails. Neste artigo, criaremos o *BD* utilizando a ferramenta IBExpert em sua versão *Standard*. Por isso, acesse o link *www.ibexpert.com* e em seguida entre no item *IBExpert*.

À esquerda do site do fabricante clique em *Download>Free*. Preencha o formulário de cadastro e aguarde o e-mail com as instruções de *download* da ferramenta. Após isso, abra o IBExpert e vamos criar o banco de dados usando a opção *Script Executive* presente no menu *Tools*.

Com a tela de *scripts* aberta, digite o código da **Listagem 2**. Nele estão contidos os esquemas para a criação do banco de dados assim como a tabela *Clientes*. Digitado o *script*, basta executá-lo usando o botão *Run Script* ou pressione a tecla *F9*.

> **Nota:** Substitua *<Caminho>\MalaDireta.fdb* pelo diretório do banco de dados e seu nome, ex: *C:\Inetpub\wwwroot\DelphiMag\MalaDireta.fdb*.

Note que inseri no *Script* alguns clientes para que possamos testar, porém os respectivos endereços eletrônicos são fictícios. Se preferir altere os dados incluindo e-mails válidos para o envio.

Enviando e-mail

O envio de e-mail é um recurso usado com freqüência em aplicações de internet. O exemplo criado nesse artigo não só envia um e-mail como permite ao internauta selecionar, em uma lista de uma tabela do banco de dados, os clientes aos quais ele deseja enviar o e-mail. Com isso, é possível criar um mecanismo de envio direcionado de mensagens.

Para que o exemplo funcione corretamente, será necessário criarmos um banco de dados e nele uma tabela chamada *Clientes*, na qual os campos *Cliente* e *Email* contenham alguns dados válidos para testes, ou seja, nomes e e-mails válidos assim como fizemos na sessão anterior. Em seguida, adicione um novo formulário em *File|New|Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Pages* chamado-o de "Gerencia Email.aspx" e que contenha os arquivos *cabecalho.ascx* e *rodape.ascx*.

Entre os controles *User Control* digite os textos "Gerenciador de envio de e-mails – ClubeDelphi – EasyDelphi" e "Clientes". Inclua um componente *CheckBoxList*("CklClientes") configurando sua propriedade *RepeatColumns* com o valor 2. Em seguida arraste dois componentes *TextBox* da paleta *Web Controls* e modifique a propriedade *ID* de cada uma para "edtAssunto" e "edtMensagem", respectivamente.

No *edtMensagem* modifique sua propriedade *TextMode* para *Multline*. Essa propriedade garante que possamos digitar mais de uma linha no *TextBox*, ou seja, nossa mensagem. Acima de cada *TextBox* digite os textos "Assunto" e "Mensagem" para que funcionem como um *Label* ("rótulo") do campo.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como baixar e instalar o IBExpert em sua versão gratuita.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

Listagem 2. Criação do banco e tabela de clientes

```
SET SQL DIALECT 3;
SET NAMES WIN1252;
SET CLIENTLIB 'C:\Program Files\Firebird\Firebird_2_0\bin\fbclient.dll';
CREATE DATABASE '<Caminho>\MalaDireta.fdb'
USER 'SYSDBA'
PAGE_SIZE 8192
DEFAULT CHARACTER SET WIN1252;
CREATE TABLE CLIENTES (
    CLIENTE  VARCHAR(30),
    EMAIL    VARCHAR(60)
);
INSERT INTO CLIENTES (CLIENTE, EMAIL) VALUES ('Jose da Silva', 'josedasilva@provedor.com.br');
INSERT INTO CLIENTES (CLIENTE, EMAIL) VALUES ('Pâmela Anderson', 'p.anderson@uol.com.br');
INSERT INTO CLIENTES (CLIENTE, EMAIL) VALUES ('Rafael Munis', 'munis@terra.com.br');
INSERT INTO CLIENTES (CLIENTE, EMAIL) VALUES ('João Morais', 'j.morais@uol.com.br');
INSERT INTO CLIENTES (CLIENTE, EMAIL) VALUES ('Marcela Almeida', 'maralmeida@ig.com.br');

COMMIT WORK;
```

Figura 5. Execução da página Session.aspx

Listagem 3. Carregamento dos clientes do banco MalaDireta.fdb

```
procedure TWebForm2.Page_Load(sender: System.Object; e: System.EventArgs);
const
  Conexao = 'User=SYSDBA;Password=masterkey; Database=<Caminho>\MalaDireta.fdb;';
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
begin
  try
    if Session.Item['Usuario'].ToString.Trim = '' then
      Response.Redirect('Session.aspx');
  except
    Response.Redirect('Session.aspx');
  end;
  if not Page.IsPostBack Then
  begin
    Conn := FbConnection.Create;
    DataAdapter := FbDataAdapter.Create;
    Comand := FbCommand.Create;
    Conn.ConnectionString := Conexao;
    Conn.Open;
    DataAdapter.SelectCommand := Comand;
    DataAdapter.SelectCommand.Connection := Conn;
    DataAdapter.SelectCommand.CommandText := 'Select * from Clientes';
    Ds := DataSet.Create;
    DataAdapter.Fill(Ds, 'Clientes');
    try
      { Configura o objet CheckBoxList }
      with CklClientes do
      begin
        DataTextField := 'Cliente';
        DataValueField := 'Cliente';
        DataSource := Ds;
        CklClientes.DataBind;
      end;
    finally
      Conn.Close();
    end;
  end;
end;
```

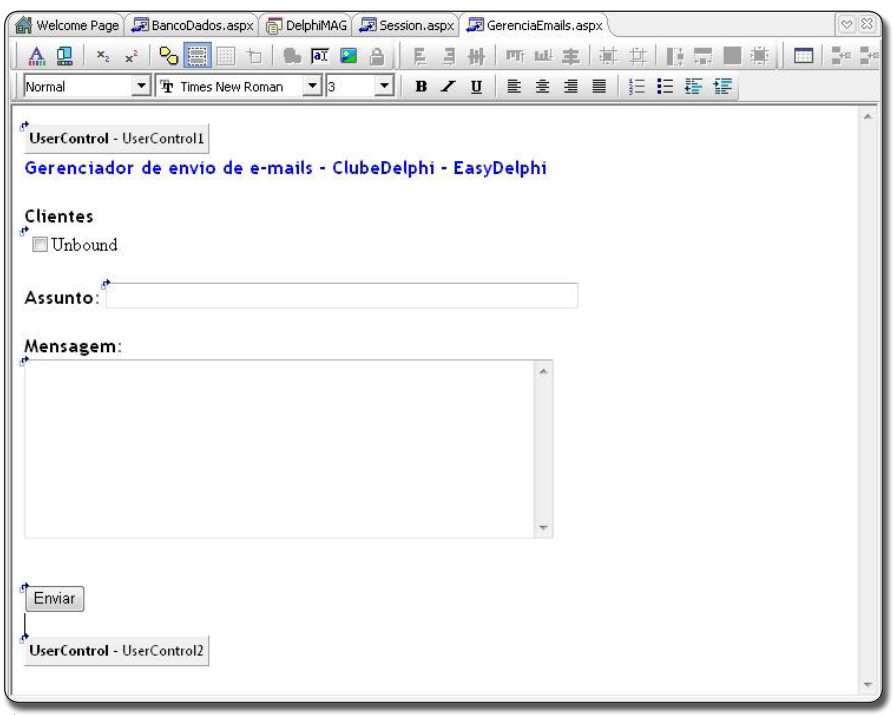


Figura 6. Execução da página de e-mail

Por último inclua um *Button*("btn Enviar") e *Text* igual a "Enviar". Esse botão fará o trabalho de selecionar os clientes e enviar as mensagens. Veja na **Figura 6** uma sugestão de layout.

Digite os códigos conforme a **Listagem 3** no *Page_Load* do formulário *Gerencia Emails.aspx*. O código consiste em checar se a sessão *Usuário* está preenchida. Caso não esteja redirecionamos o usuário para a página *Session.aspx* usando o método *Redirect* do objeto *Response*, como podemos ver a seguir:

```
try
  if Session.Item['Usuario'].ToString.
    Trim = '' then
      Response.Redirect('Session.aspx');
except
  Response.Redirect('Session.aspx');
end;
```

Note que incluímos a verificação em um bloco *try..except*, desta forma se houver qualquer erro durante o carregamento da página, automaticamente o usuário será novamente redirecionado para *Session.aspx*. Essas ações proporcionam segurança ao site e evitam que uma página seja aberta diretamente na *URL*, ou seja, digitando o endereço na barra de endereços do *browser*. Você deve aplicar essa técnica a todos os formulários existentes que requeiram *login*.

A parte 2 de nosso treinamento mostrou o acesso a banco de dados. Usamos a mesma técnica empregada no artigo anterior para listar os dados da tabela *Clientes* e preencher o componente *CheckBoxList*. Como usaremos a mesma conexão com o banco de dados em mais de um lugar, declare a constante *Conexão* abaixo da palavra *Implementation* do formulário como fizemos no artigo anterior. Essa constante deve receber a *String* de conexão. Veja em seguida um exemplo:

```
const
  Conexao = 'User=SYSDBA;Password=masterkey;
    Database=<Caminho>\MalaDireta.fdb';
```

Nota: Não esqueça de declarar os *namespaces FirebirdSql.Data.Firebird, System.Text, Borland.Vcl.SysUtils e System.Web.Mail*. Esses *namespaces* contêm a declaração das principais funções/procedimentos que usamos nessa página tais como: conexão com o *Firebird*, envio de e-mails e criação de *StringBuilder*.

Assim que a página *GerenciaEmail* for aberta, ela aparecerá montada e exibirá a lista de clientes cadastrados na tabela *Clientes* no banco de dados. Selecione os clientes que irão receber o e-mail e digite o assunto e a mensagem. Para disparar o e-mail, clique no botão *Enviar*. Caso queira melhorar este exemplo, você pode criar grupos de clientes organizados por estados, categorias etc. e disparar e-mails para um determinado grupo. A **Listagem 4** deve ser digitada no evento *Click* do botão enviar.

No código da **Listagem 4** estamos envolvendo o objeto *CklClientes* em um laço *for* onde capturamos os clientes que o usuário selecionou. Nesse *for* estamos criando uma *String* contendo a cláusula *In* que será usada na *Select* de pesquisa dos clientes. Simplificando, estamos selecionando no banco de dados todos os clientes marcados no componente *CklClientes* por meio de uma instrução SQL criada em tempo de execução.

Em seguida montamos um novo *for*, desta vez envolvendo um objeto do tipo *DataTable*. Esse objeto é criado e recebe a tabela que será usada para enviar mensagens aos clientes. Como segue:

```
Tabela := Ds.Tables.Item['Clientes'];
```

Por fim verificamos a existência de dados nos campos *Assunto* e *Mensagem*, montamos o e-mail na variável de Mensagem do tipo *MailMessage* e enviamos chamando o método *Send*.

```
From := 'voce@empresa.com.br';
Mensagem.&To := Tabela.Rows[I]['Email'].
  ToString;
Subject := edtAssunto.Text.Trim.ToString;
Body := edtMensagem.Text.Trim.ToString;
SmtpMail.Send(Mensagem);
```

Criando arquivos XML

XML (*Extensible Markup Language* – Linguagem de Marcação Extensível) consiste em um padrão mundial que tem conquistado o mercado em aplicativos de integração de dados entre plataformas, aplicações e fontes de dados. Neste exemplo, veremos como ler uma tabela do banco de dados, gerar um *Dataset* e gravar um arquivo XML. Por outro lado, caso você receba um arquivo XML de outra empresa ou aplicação, precisa saber como ler o respectivo arquivo.

Adicione um novo formulário chamado "XML.aspx" que contenha o cabeçalho e o rodapé existentes nos arquivos .ascx. O layout desta página pode ser visto na **Figura 7**.

Insira dois botões cujo *IDs* são "btnGravar" e "btnLer", respectivamente. Abaixo inclua um *DataGrid*("grdXML") e escolha sua formatação preferida usando o botão direito e *Auto Format*. Por último arraste um *Label*("lblMensagem").

Para criar um arquivo XML digite o código da **Listagem 5** no evento *Click* do *btnGravar*. A criação do arquivo XML consiste em conectar-se ao banco de dados, fazer uma consulta e alimentar um *DataSet*. Nomeamos esse *DataSet* como *dsXML*, ele receberá o resultado da consulta no banco de dados. Em seguida utilizamos o método *WriteXML* do *DataSet* informando qual endereço

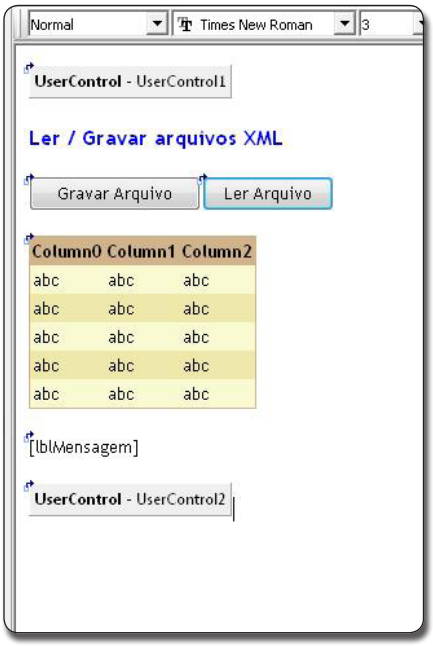


Figura 7. Layout da página XML.aspx

Listagem 4. Código para envio de e-mail

```
procedure TWebForm2.btnEnviar_Click(sender: System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
  DR : FbDataReader;
  Condicao : StringBuilder;
  Selecao : System.Text.StringBuilder;
  I : Integer;
  Tabela : DataTable;
  Mensagem: MailMessage;
begin
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  Conn.ConnectionString := Conexao;
  Conn.Open;
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  Selecao := System.Text.StringBuilder.Create;
  Selecao.Append('Select * from Clientes');
  Condicao := System.Text.StringBuilder.Create;
  Condicao.Append(' Where Cliente In (');
  for I := 0 to CklClientes.Items.Count-1 do
    if CklClientes.Items[I].Selected then
      { Para usar o QuotedStr no ASP.NET declare o namespace Borland.Vcl.SysUtils }
      Condicao.Append(QuotedStr(CklClientes.Items[I].Value) + ',');
  Condicao.Append(')');
  Condicao := Condicao.Replace(',)',')');
  Selecao.Append(Condicao);
  Comand.CommandText := Selecao.ToString;
  Ds := DataSet.Create;
  DataAdapter.Fill(Ds, 'Clientes');
  Tabela := Ds.Tables.Item['Clientes'];
  Mensagem := MailMessage.Create;
  { Exemplo smtp.uol.com.br }
  SmtpMail.SmtpServer := 'smtp.empresa.com.br';
  for I := 0 to Tabela.Rows.Count - 1 do
  begin
    if not ((edtAssunto.Text.Trim = '') or (edtMensagem.Text.Trim = '')) then
    begin
      with Mensagem do
      begin
        From := 'voce@empresa.com.br';
        Mensagem.&To := Tabela.Rows[I]['Email'].ToString;
        Subject := edtAssunto.Text.Trim.ToString;
        Body := edtMensagem.Text.Trim.ToString;
        SmtpMail.Send(Mensagem);
      end;
    end;
  end;
  Mensagem.Free;
end;
```

Listagem 5. Códigos para gravar o arquivo XML

```
procedure TWebForm2.btnGravar_Click(sender: System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  DataAdapter : FbDataAdapter;
  Ds : DataSet;
  Comand : FbCommand;
  dsXML : DataSet;
begin
  Conn := FbConnection.Create;
  DataAdapter := FbDataAdapter.Create;
  Comand := FbCommand.Create;
  Conn.ConnectionString := Conexao;
  Conn.Open;
  DataAdapter.SelectCommand := Comand;
  DataAdapter.SelectCommand.Connection := Conn;
  DataAdapter.SelectCommand.CommandText := 'Select * from Clientes';
  Ds := DataSet.Create;
  dsXML := DataSet.Create;
  DataAdapter.Fill(dsXML, 'Clientes');
  gdrXML.Visible := False;
  lblMensagem.Visible := True;
  try
    dsXML.WriteXml('C:\Inetpub\wwwroot\DelphiMag\XML.xml');
    lblMensagem.Text := 'Arquivo gerado com sucesso';
  finally
    Conn.Close;
  end;
end;
```

Listagem 6. Código para ler o arquivo XML

```
procedure TWebForm2.btnLer_Click(sender: System.Object; e: System.EventArgs);
var
  dsXML : DataSet;
begin
  Try
    dsXML := DataSet.Create;
    dsXML.ReadXml('C:\Inetpub\wwwroot\DelphiMag\XML.xml');
    gdrXML.Visible := True;
    gdrXML.DataSource := dsXML;
    gdrXML.DataBind;
    lblMensagem.Visible := True;
    lblMensagem.Text := 'Existem ' + dsXML.Tables[0].Rows.Count.ToString +
     ' registros';
  except
    lblMensagem.Text := 'Ocorreu um erro na leitura';
    gdrXML.Visible := False;
  end;
end;
```

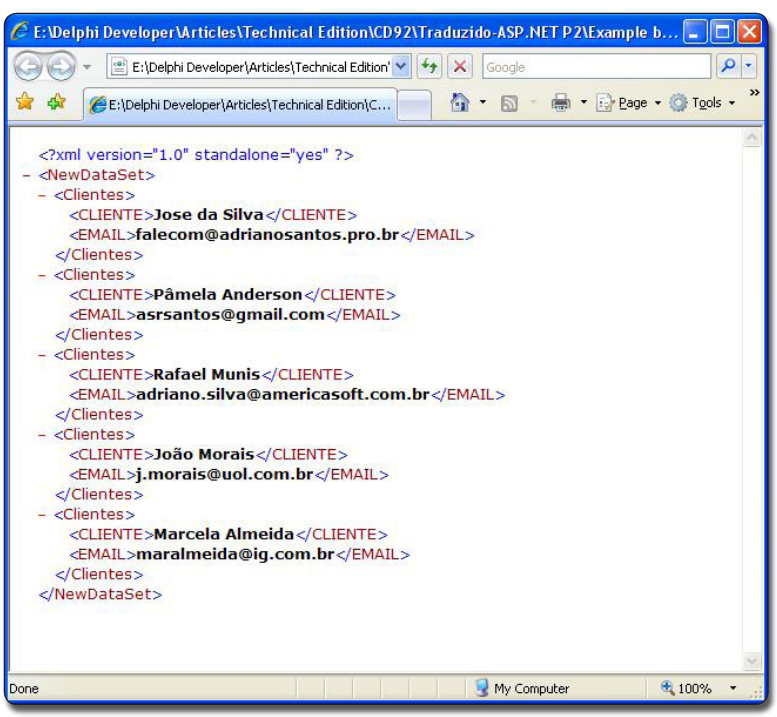


Figura 8. Arquivo XML aberto no navegador

que o arquivo deve ser salvo, nesse caso em *C:\Inetpub\wwwroot\DelphiMag*, ou seja, o diretório de nossa aplicação, como podemos ver em seguida.

```
dsXML.WriteXml('C:\Inetpub\wwwroot\
DelphiMag\XML.xml');
```

A leitura do XML é ainda mais simples. Na **Listagem 6** podemos ver que apenas criamos um *DataSet* e chamamos seu método *ReadXml* indicando o caminho do arquivo. Automaticamente o *DataSet* é preechido, agora basta associarmos o *dsXML* a propriedade *DataSource* do *grdXML* e chamar o método *DataBind*. O restante são atualizações de controles visuais.

As últimas pronvidências a se tomar são: declarar o namespace *FirebirdSql.Data.Firebird* na sessão *Uses* do formulário e incluir verificação da *Session* no *Page_Load* como segue:

```
try
  if Session.Item['Usuario'].ToString.
  Trim = '' then
    Response.Redirect('Session.aspx');
except
  Response.Redirect('Session.aspx');
end;
```

A **Figura 8** mostra o arquivo XML gerado pelo código criado no botão Gravar da página XML.aspx.

Conclusão

Neste artigo finalizamos o treinamento em ASP.NET vendo como fazer uso de *Session* em nossas aplicações. O uso de *Session* pode ser útil na programação dos formulários isolando os dados de determinadas situações de outras seções. Um exemplo prático de uso são os *carrinhos de compras* em lojas virtuais.

O uso de e-mails em aplicações de marketing, vendas, CRM etc., é fundamental para o relacionamento com os clientes. O código apresentado neste artigo demonstrou o uso real deste recurso, que pode ser implementado em qualquer aplicação na empresa. Por fim vimos também como fazer uso dos *User Controls* para padronizar cabeçalho e rodapé.

Espero que, com esta série de artigos sobre o ASP.NET, você consiga desenvolver aplicações para WEB e colocar os dados de sua empresa ao alcance de todos os interessados. *"No stress, think .NET"*. ●

Um bom software também precisa de um bom layout!

Kameha
webdesign

Vídeo-aulas especializadas em Design Digital.

- Aprenda através de vídeo-aulas 100% práticas, tudo que as ferramentas de design podem lhe oferecer. Desde o básico ao mais avançado recurso.
- Assine agora mesmo e receba:

+ de 50 vídeo-aulas de acesso imediato

+ de 10 vídeos inéditos por mês.

Por apenas 6x de **R$ 13,61**

Estas são algumas das ferramentas que já estão disponíveis para você:

PhotoShop	Corel Draw
Flash	After Effects
DreamWeaver	e muito mais...

Acesse e assine já!

www.kameha.com.br

Um produto da DevMedia GROUP

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Delphi for PHP

Crie um site de publicação de notícias – Parte 5

Bem chegamos ao final do nosso mini-curso e vamos fazer a publicação da aplicação. Para testar na prática a aplicação conseguimos um provedor que nos cedeu o espaço e as tecnologias necessárias.

O processo não é difícil, mas um bom provedor facilitará bastante a publicação e o gerenciamento da mesma. Pense nisso na hora de escolher o seu.

Verificando as configurações

Antes de publicar a aplicação é necessário saber os requisitos mínimos para o funcionamento. O recomendado é seguir as mesmas instalações e configurações que o Delphi for PHP utiliza. Para verificar essas configurações na sua máquina crie uma nova aplicação no Delphi for PHP, adicione um componente *Label* ao formulário e no evento *OnBeforeShow* do formulário coloque o seguinte código:

```
$this->Label1->Caption=phpinfo();
```

Compile a aplicação e veja uma página com informações sobre o PHP que você está utilizando. Essa é a finalidade da função *phpinfo*. Com essa página é possível saber a versão corrente do PHP, quais módulos estão carregados e algumas variáveis importantes como *post_max_size* que define o tamanho máximo de um *post*, como, por exemplo, o envio de fotos em nosso cadastro (**Figura 1**). Sabendo as configurações que temos em nossa máquina teremos que analisar as configurações do servidor no provedor.

O Provedor

Para exemplificar a publicação de nossa aplicação a DevMedia escolheu o serviço de hospedagem da Hospedix, que possibilita um plano experimental de 30 dias para assinantes da ClubeDelphi.

Realizei contato com a equipe técnica que disponibilizou o endereço *minicurso.devmedia.com.br*, usuário e senha para acesso ao painel de controle e informou os passos restantes para configuração

Fabrício Desbessel
(fabricio@fabricio.pro.br)
é professor de Linguagem de Programação do Curso Técnico em Informática do Colégio Frederico Jorge Logemann de Horizontina/RS e da FAHOR Faculdade Horizontina. Delphiano de coração está sempre disposto a provar que com o Delphi sempre teremos a melhor solução. Site www.fabricio.pro.br.

das linguagens utilizadas, *FTP* para envio dos arquivos e criação da base de dados. De posse do usuário e senha realizou-se o acesso ao site da Hospedix e logou-se no Painel de Controle, o qual mostrou-se bem intuitivo e de boa usabilidade. (**Figura 2**)

Como o domínio já estava criado, passou-se para a instalação da aplicação, através do link *Install a Site application*. Na primeira página escolhe-se qual dos domínios será configurado. Na próxima página define-se o *login* e senha para *FTP*, bem como os serviços disponíveis. Como serviço habilitou-se o PHP na versão 5.2.4, conforme mostra a **Figura 3**.

Na página seguinte são oferecidas algumas aplicações padrão que poderão ser utilizadas no site como gerenciadores de conteúdos, *chat*, fórum e serviço de *webmail*. Clicou-se em *Cancelar*, pois não é o objetivo do artigo instalar essas aplicações.

Com essas configurações pode-se acessar o *FTP* e enviar um arquivo PHP para teste. Criou-se esse arquivo com o seguinte código:

```
<?php phpinfo(); ?>
```

O arquivo foi salvo com o nome "teste.php" e enviado para dentro da pasta *httpdocs* do *FTP*. Pelo *browser* foi possível executar o teste e ter as informações do PHP do servidor, conforme exibido na **Figura 4**.

Com o resultado apresentado na **Figura 4**, tem-se a certeza que a aplicação criada no Delphi for PHP irá funcionar, pois o servidor utiliza uma versão superior à mínima exigida. Mesmo assim, ainda pode-se ter alguns problemas referentes a algumas bibliotecas, que no PHP são carregadas através de módulos, como é o caso do *mbstring*.

Criando o Banco de Dados

Para criar o banco de dados novamente acessou-se o painel de controle da Hospedix, clicando no link *Create a database*. Escolheu-se o domínio e passou-se para a próxima página. Na página seguinte definiu-se o nome da base de dados e servidor que será utilizado, conforme mostra a **Figura 5**.

Finalizando, criou-se um banco de

PHP Version 5.1.3

System	Windows NT NOTFABRICIO 5.1 build 2600
Build Date	May 1 2006 00:27:04
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Arquivos de programas\CodeGear\Delphi for PHP\1.0\php\php.ini
PHP API	20041225
PHP Extension	20050922
Zend Extension	220051025
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, http, ftp, compress.zlib
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.1.0, Copyright (c) 1998-2006 Zend Technologies
with DBG v2.13.1, (C) 2000,2006, by Dmitri Dmitrienko

Powered By Zend Engine 2

Figura 1. Resultado da execução da função phpinfo() no Delphi for PHP

Figura 2. Parte do painel de controle da Hospedix

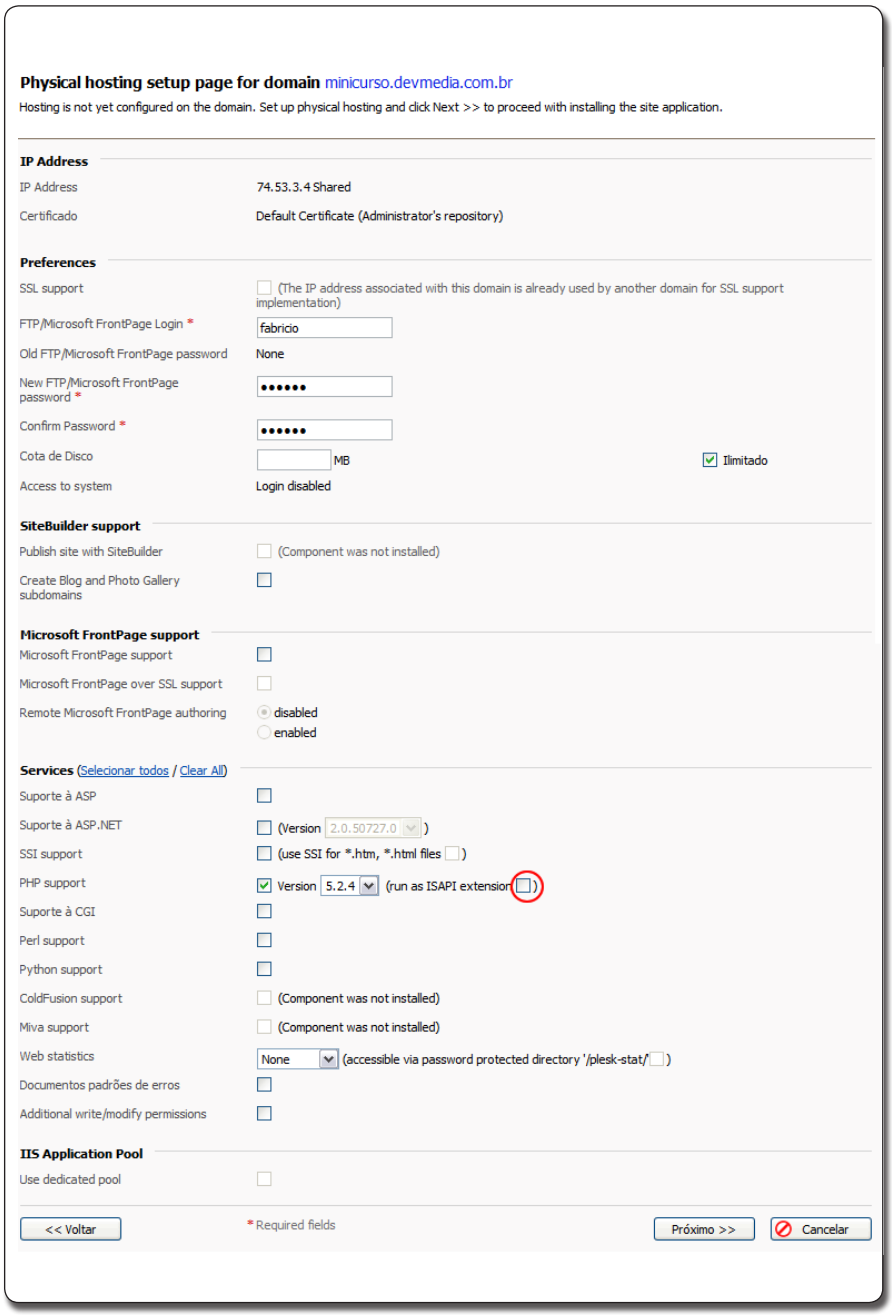


Figura 3. Configurações do site no Painel de Controle

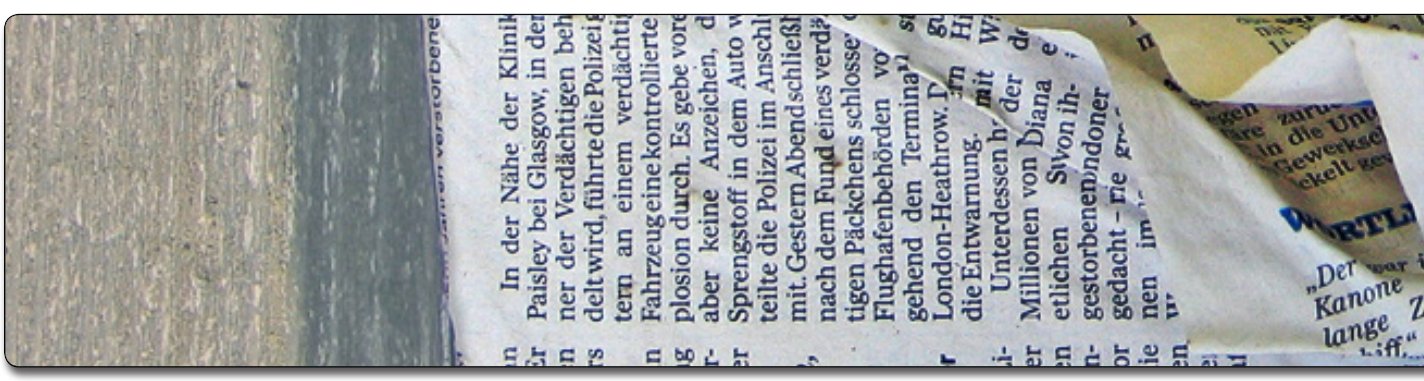
dados chamado *delphiphp_jornal*, usando o servidor MySQL. Para criar o usuário do banco e também executar o *script* de criação das tabelas, necessitou-se exibir todas as opções do painel de controle para o site. Para fazer isso, clicou-se no link *Home* (à esquerda, no topo) e depois selecionou-se o site. Com isso é apresentado o painel completo, com todas as opções, conforme mostra a **Figura 6**.

Continuando na configuração do banco de dados, clicou-se no link *Bases de Dados*, onde selecionou-se a base de dados *delphiphp_jornal*, criada anteriormente. Nesse momento, criou-se um usuário e senha para acesso ao banco. Depois de criar o usuário foi disponibilizado um ícone com o nome *WebAdmin*, que abre a *interface* do *phpMyAdmin*, um gerenciador muito conhecido no mundo PHP + MySQL.

No *phpMyAdmin* selecionou-se o banco *delphiphp_jornal* e na página seguinte clicou-se no link *Importar*. Procurou-se o arquivo do *script* do banco, criado na primeira parte do mini-curso e executou-se. Com isso as tabelas foram criadas e possibilitou-se a inserção de dados na tabela de usuários para poder fazer o primeiro *login* na aplicação. O *phpMyAdmin* é bem explicativo, está em português e é muito fácil de utilizar. Assim finalizamos a criação do banco de dados e podemos agora publicar a aplicação e fazer os primeiros testes

Antes do Deploy

Abra o projeto no Delphi for PHP e, primeiramente acerte as propriedades *DatabaseName*, *UserName* e *UserPassword* do componente *Database*, existente no *DataModule*, de acordo com o banco,



usuário e senha criados no servidor onde será publicada a aplicação.

Também renomeie o arquivo do *template* que criamos na parte anterior do curso, modificando seu nome para "modelo.html". Com isso será necessário abrir a página *index.php* e alterar a propriedade *TemplateFilename* informando *modelo.html*. Essa alteração é necessária para que essa página de modelo não seja a exibida quando alguém acessar o endereço *minicurso.devmedia.com.br*.

Nas páginas que exibem fotos, deve-se alterar os eventos *OnBeforeShow* dos componentes *Image's*, mais especificamente a linha que define a variável diretório (*$diretorio*), para que seja possível detectar o diretório do servidor, através da variável de ambiente *$_SERVER["DOCUMENT_ROOT"]*, que possui essa função. Então verifique esses eventos em todas as páginas e troque o texto "C:/CD/Jornal/" por *$_SERVER["DOCUMENT_ROOT"]*.

Na página *login.php* é necessário limpar os campos de usuário e senha a cada vez que a página é chamada. Para isso no evento *OnBeforeShow* desse formulário acrescente mais essas duas linhas:

```
$this->edtUsuario->Text='';
$this->edtSenha->Text='';
```

Fazendo o Deploy

No menu *Tools* acesse a opção *Deployment Wizard*. Com esse assistente fica fácil separar os arquivos que deverão ser enviados ao servidor. Na primeira tela simplesmente clique em *Next* e na segunda tela defina o lugar (diretório) onde serão colocados os arquivos. O processo de *delpoy* é bem simples e começa com a verificação dos arquivos do projeto (.php), os componentes existentes

PHP Version 5.2.4

System	Windows NT HSPDX-SHARED-01 5.2 build 3790
Build Date	Aug 30 2007 07:05:48
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared"
Server API	CGI/FastCGI
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\Program Files\SWsoft\Plesk\Additional\PleskPHP5\php.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	enabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, data, http, ftp, compress.zlib
Registered Stream Socket Transports	tcp, udp
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, zlib.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.2.0, Copyright (c) 1998-2007 Zend Technologies

Powered By Zend Engine 2

Figura 4. Resultado da execução da função phpinfo() servidor Hospedix

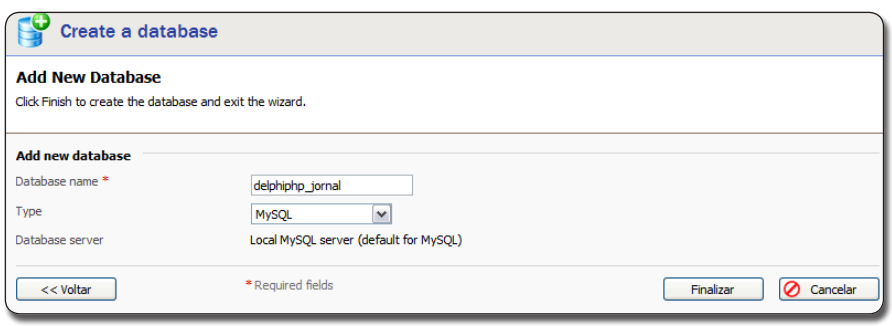


Figura 5. Configuração do Banco de Dados no Painel de Controle

nesses arquivos e as classes necessárias para cada componente. Para ter um detalhamento desse processo pode-se marcar a opção *Detailed process*. No final, clique no botão *Finish*.

Porém, infelizmente o *Deploy* não se preocupa com outros arquivos que adicionamos ao projeto, como as imagens e a página *modelo.html*, utilizada no *template*. Portanto é necessário copiá-los para o mesmo diretório do *Deploy*. Como o projeto prevê um diretório para armazenar as fotos também é necessário criar esse diretório dentro do *Deploy*.

Com isso finalizamos o *Deploy* e estamos prontos para publicar a aplicação. Basta conectar no *FTP* e copiar todos os arquivos e diretório para a pasta *httpdocs*.

Fica muito difícil fazer o *upload* dos arquivos conectando ao *FTP* pelo *browser*. Aconselho a utilização de um software específico para envio de arquivos via *FTP* que tenha como característica, a capacidade de reconectar ao servidor se a conexão for finalizada. No meu caso, utilizei o software *Core FTP LE*, exibido na **Figura 7**.

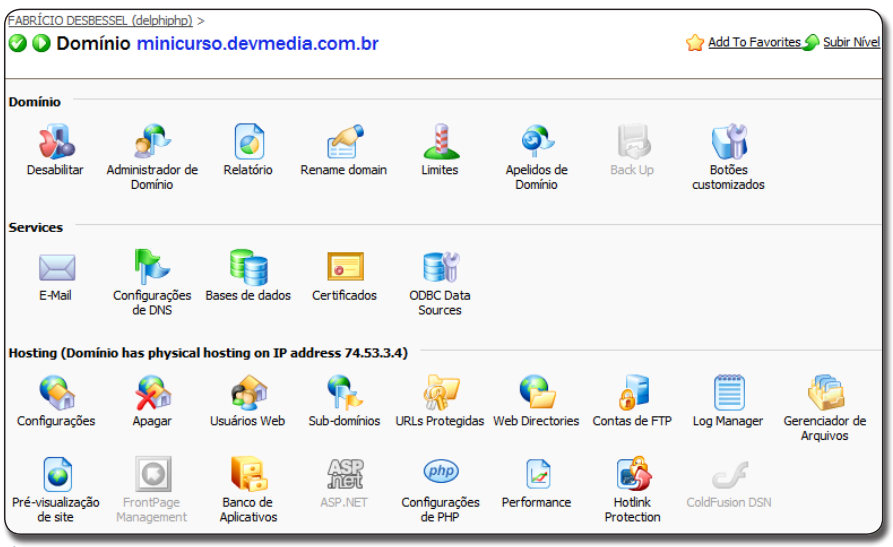


Figura 6. Todas as opções de configuração para o site no Painel de Controle

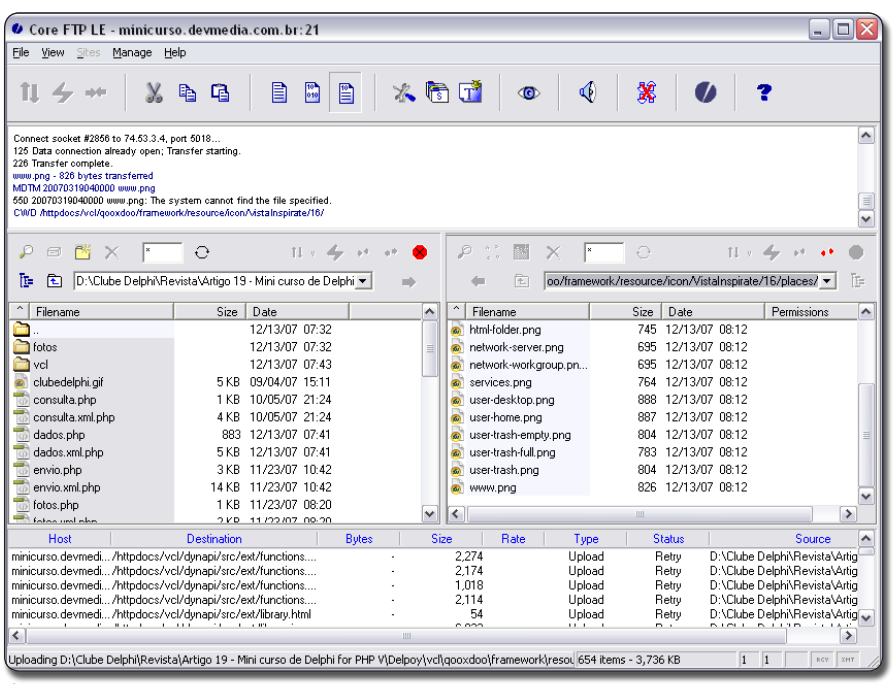


Figura 7. Software para upload da aplicação por FTP

Testando a aplicação

Ao tentar executar a aplicação chamando a página *index.php* obtive o erro da falta da classe *Smarty*, que é a classe responsável pela utilização de *templates*. Realmente o *Deploy* esqueceu dessa classe. Então, manualmente copie o diretório *smarty* que encontra-se dentro da pasta *vcl*, da instalação do Delphi for PHP para a pasta *Deploy* e faça novamente o *upload* por *FTP*. Também obtive a seguinte mensagem de erro:

```
Warning: unserialize() expects parameter
1 to be string, array given in D:\web\
minicurso.devmedia.com.br\httpdocs\jornal\
vcl\dbtables.inc.php on line 1017.
```

Poderia ser realizada a atualização de toda a *VCL* através do site: *www.qadram.com/vcl4php/*. Mas não recomendo, pois os códigos poderão precisar de alterações, ou seja, teria que ser revisada toda a aplicação. Procurando na internet encontrei uma solução em um fórum que solicita que se altere a função *WriteSQL* no arquivo *dbtables.inc.php*. Então substitua a função pela seguinte:

```
function writeSQL($value) {
  $this->_sql=$value;
}
```

O módulo *mbstring* na maioria dos provedores não está carregado. É interessante solicitar ao provedor o carregamento desse módulo que define padrões de caracteres de entrada e saída *http*, resolvendo muito problemas de acentuação.

Também aconteceu o erro "*The Input Filter PHP extension is not setup on this PHP installation*", comentado pela CodeGear no artigo *support.codegear.com/article/36378*. Para resolver isso tem-se duas opções. A primeira é fazer a alteração do arquivo *system.inc.php*, desabilitando a exceção de erro, conforme comenta o artigo e a segunda opção é utilizar o arquivo *system.inc.php* mais recente, baixando a atualização da *VCL*.

Outra mensagem de erro encontrada que dizia: Error *in my_thread_global_end(): 1 threads didn't exit*, mereceu uma atenção maior. Esse erro não se referia a nossa aplicação, pois aparecia também executando a página *teste.php*. Procurando na internet descobri que se

trata de um *bug* de uma biblioteca do MySQL e que ainda não foi solucionado. Mesmo assim, na Hospedix é possível contornar a situação passando a utilizar o PHP via *ISAPI*. Para isso, no painel de controle acesse as configurações do domínio e marque a opção *run as ISAPI extension*, conforme é exibido no círculo da **Figura 3**. Claro que utilizar PHP via *ISAPI* não é a melhor alternativa, mas enquanto o *bug* não é corrigido essa é a única solução. Com isso a aplicação deverá rodar perfeitamente.

Correções

Necessitei fazer uma correção na página *login.php* pois a mesma não apresentava mensagem de erro de usuário e senha incorretos. Portanto, abra essa página no Delphi for PHP e substitua o código do botão *Logar* pelo código da **Listagem 1**. Para um melhor entendimento, todas as listagens foram comentadas.

Também foi necessário adaptar o formulário de envio do e-mail para utilizar a classe oferecida pela Hospedix que é a *PHPMailer*. Então substitua o código do botão *Enviar* pelo código da **Listagem 2**.

Conclusão

Com isso finalizamos nosso mini-curso de Delphi For PHP com banco de dados MySQL. Espero que vocês tenham gostado de utilizar o Delphi for PHP assim como eu. Acredito que essa ferramenta vai crescer e se tornar uma das mais utilizadas para trabalhar com PHP.

Para quem quiser acessar o site e ver na prática seu funcionamento, a Hospedix vai mantê-lo por alguns meses. Para acessar utilize o link *minicurso.devmedia.com.br* e para entrar na parte administrativa utilize o usuário *assinante@clubedelphi.net* e senha *devmedia*.

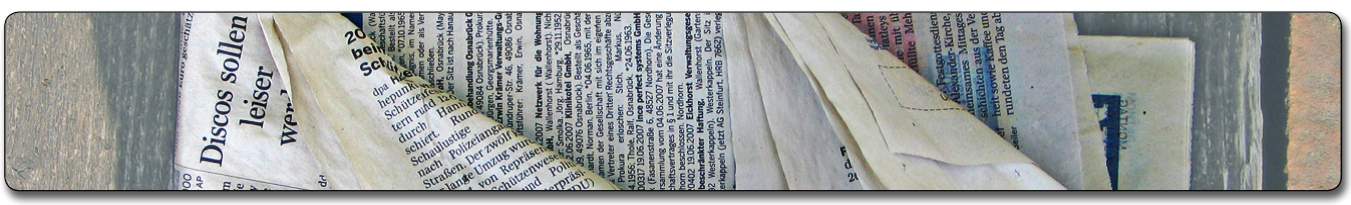
Se restaram dúvidas ou problemas que não foram tratados nos artigos, enviem para meu e-mail que poderei responder nas seções de perguntas e respostas da revista. ●

Listagem 1. Código do botão logar

```
$this->sqlUsuarios->Close();
$this->sqlUsuarios->Filter="email='".addslashes($this->edtUsuario->Text).
"' AND senha ='".addslashes($this->edtSenha->Text)."'";
$this->sqlUsuarios->Open();
if ($this->sqlUsuarios->RecordCount > 0) {
   $_SESSION['id_usuario'] = $this->sqlUsuarios->id_usuario;
   $_SESSION['nome'] = $this->sqlUsuarios->nome;
   $_SESSION['nivel'] = $this->sqlUsuarios->nivel;
   if ($this->lblUrl->Caption != NULL){
        redirect($this->lblUrl->Caption);
   } else {
        redirect('principal.php');
   }
   $this->sqlUsuarios->Close();
} else {
   $this->sqlUsuarios->Close();
   redirect('login.php?erro=Usuário ou senha incorreto!');
}
$this->sqlUsuarios->Close();
```

Listagem 2. Código do botão Enviar

```
$server = "http://minicurso.devmedia.com.br";
$comment = $this->memComentario->Text;
$link = $server . "index.php?id_noticia=" . $this->sqlNoticia->id_noticia;
$from = $this->edtSeuNome->Text;
$message = "$from achou que a noticia abaixo seria interessante para você.\n\n" .
$this->sqlNoticia->titulo . "\n\nPara acessá-la clique no link abaixo:\n" . $link;
if ($comment != "")
{
   $message .= "\n\nComentário:\n$comment";
}
require("class.phpmailer.php");
$mail = new PHPMailer();
$mail->IsSMTP();
// Servidor de SMTP
$mail->Host = "mail.hospedix.com.br";
// SMTP com autenticação
$mail->SMTPAuth = true;
// Entre com os dados de sua conta de e-mail na Hospedix para autenticação em nossos servidores.
// SMTP com autenticação
$mail->Username = "LOGIN FORNECIDO PELO HOSPEDIX";
// Senha do e-mail da linha acima
$mail->Password = "SENHA DO LOGIN";
//Devido a nossa política de segurança que visa manter a qualidade dos serviços
  oferecidos aos nossos assinantes, o rementente deverá ser um e-mail do seu domínio
  na Hospedix.
// Sua conta de e-mail na Hospedix
$mail->From = " LOGIN FORNECIDO PELO HOSPEDIX ";
// Nome de exibição do remetente
$mail->FromName = $this->edtSeuNome->Text;
//E-mail de Reposta do Remetente
$mail->AddReplyTo($this->edtSeuEmail->Text, "");
// Seu e-mail de destino
$mail->AddAddress("LOGIN FORNECIDO PELO HOSPEDIX ", "");
// Seu e-mail de destino
$mail->IsHTML(true);
// Assunto do e-mail
$mail->Subject = "Notícia Interessante";
// Corpo do e-mail
$mail->Body = $message;
$mail->AltBody = " ";
if($mail->Send())
{
   $msg = "Notícia enviada com sucesso!<br><br>Obrigado!";
   global $UDados;
   $Atual = $this->sqlNoticia->qtde_envio;
   $Atual = $Atual + 1;
   $UDados->dbjornal1->execute("Update noticias Set qtde_envio=" .
   $Atual . " Where id_noticia=" .
   $this->sqlNoticia->id_noticia);
}
else
{
   $msg = "Erro Interno. Tente novamente mais tarde.";
}
$this->lblMensagem->Caption = $msg;
redirect('envio.php?id_noticia=?' . $this->sqlNoticia->id_noticia);
```



Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Introdução ao PHP – Parte 4

Como criar variáveis globais e estáticas e transferir valores entre páginas

Ewald Geschwinde e
Hans-Juergen Schoenig

Essa é a última parte de nosso mini-curso *Introdução ao PHP* onde estamos vendo os principais detalhes da linguagem. Neste artigo aprenderemos como trabalhar com mais de uma página PHP transferindo informações de uma página para outra através de variáveis globais e estáticas, além de formulários.

Veremos também como criar *packages* ("pacotes"), que na verdade são páginas PHP contendo variáveis e funções a serem utilizadas. Entenderemos como o PHP se comporta ao efetuar "includes" de arquivos contendo código de funções e variáveis.

Entendendo as variáveis

As variáveis não são "visíveis" na aplicação inteira. Este é um aspecto muito importante, pois, caso contrário, os programadores acabariam tendo complicações. Ao construir aplicações grandes é importante saber onde e quais variáveis podem ser vistas e utilizadas. Comecemos com um exemplo simples na **Listagem 1**.

Em primeiro lugar, *$a* é inicializada e exibida na tela. Então a função *display_data* é chamada, e tenta exibir *$a* novamente e o que temos é o resultamos a seguir:

```
Mensagem principal: Olá Mundo!
dados:
```

Como podemos observar *display_data* não consegue exibir nada, pois *$a* não está visível para ela. A variável *$a* foi inicializada no script principal e, portanto, não é visível para funções chamadas ele. Por este motivo, se desejarmos acessar *$a* em *display_data*, devemos então, declará-la como global assim como podemos ver na **Listagem 2**.

Com a ajuda da palavra-chave *global*, é possível ler e modificar qualquer variável global:

```
Mensagem principal: Olá Mundo!
dados: Olá Mundo!
Mensagem principal: Olá Adriano
```

Como podemos ver o conteúdo de *$a* pode ser modificado por *display_data*. Outro modo de acessar a variável seria utilizar um *array* chamado *$GLOBALS* como podemos ver na **Listagem 3**.

Com a ajuda de *$GLOBALS*, é possível acessar qualquer variável que foi definida globalmente. Executando o *script*, a saída será exibida na tela:

```
Mensagem Principal: Olá Mundo!
dados:Olá Mundo!
```

Outro aspecto importante a ser considerado, especialmente quando lidamos com projetos muito grandes, é que podemos incluir outros arquivos no script usando o comando *include*, desta forma variáveis hospedas em outros arquivos podem ser visualizadas.

Com a ajuda do comando *include*, os arquivos que contêm funções e variáveis são acrescentados a um *script* PHP e serão visíveis pelo interpretador. Na segunda linha da **Listagem 3** estamos incluindo o arquivo *vars.php* presente na mesma pasta onde o script atual está sendo executado.

Experimente criar um arquivo *.php* e digitar nele o código a seguir. Note que definimos uma variável chama *$a* cujo o conteúdo é "Olá Mundo!".

```
<?php
    # Defining a string
    $a = "Olá Mundo!";
?>
```

Crie um novo arquivo e insira o código da **Listagem 3** antes mencionada. Salve-o e execute-o no browser. É recomendável colocar arquivos que contêm somente definições de variáveis em um diretório separado, para que o mesmo não seja misturado com arquivos que contêm outros códigos-fonte. Além do mais aumenta a segurança do site. Executando o *script*, veremos que o resultado será o mesmo do exemplo anterior:

```
Mensagem Principal: Olá Mundo!
dados: Olá Mundo!
```

Incluir um arquivo é simples, mas o que acontece quando o arquivo incluso em *script.php* contém, por sua vez, arquivos outros arquivos inclusos? Vejamos no código a seguir uma nova abordagem. Para entendermos como o PHP se comporta nesses casos, crie dois novos arquivos PHP com os nomes "base_a.php" e "base_b.php". No primeiro arquivo, *base_a.php*, digite o código a seguir:

```
<?php
  $a = "Olá Mundo!";
?>
```

Já no arquivo *base_b.php* digite:

```
<?php
  $b = "Somos do Brasil";
?>
```

Basicamente estamos definindo, separadamente, o valor de duas variáveis: *$a* e *$b*. Em seguida modifique o script do arquivo *vars.php* conforme segue:

```
<?php
  include ("base_a.php");
  include ("base_b.php");
?>
```

Desta vez o *vars.php* não contém a definição das variáveis, mas inclui dois arquivos que contêm esta definição de *$a* e *$b*. Por fim crie um novo arquivo PHP que implementa o código a seguir:

```
<?php
  include ("vars.php");
  echo $a<br>";
  echo $b;
?>
```

Nesse caso estamos incluindo o arquivo *vars.php* que contém inclui os dois arquivos criados anteriormente que por sua vez contêm os valores das variávies *$a* e *$b*. O que acontece em seguida é que ambas variáveis são exibidas no browser. Isso acontece porque as variáveis inclusas no arquivo *vars.php* também são adicionadas nesse script. Veja a seguir:

```
Olá mundo!
Somos da América
```

Entendendo e usando variáveis estáticas

As variáveis estáticas, assim como muitos outros componentes do PHP, foram emprestadas do C/C++. Essas

Listagem 1. Exemplo de variável não visível a toda aplicação

```
<?php
  $a = "Olá Mundo!";
  echo "Mensagem principal: $a<br>\n";
  display_data();

function display_data(){
  echo "dados: $a<br>\n";
}
?>
```

Listagem 2. Declaração de variável global

```
<?php
  $a = "Olá Mundo!";
  echo "Mensagem principal: $a<br>\n";
  display_data();
  echo "Mensagem principal: $a<br>\n";

function display_data(){
  global $a;
  echo "dados: $a<br>\n";
  $a = "Olá Adriano";
}
?>
```

Listagem 3. Usando o array $GLOBALS

```
<?php
  include("inc/vars.php");
  echo "Mensagem Principal: $a<br>\n";
  display_data();

function display_data(){
  echo "dados:".$GLOBALS["a"];
}
?>
```

Listagem 4. Exemplo de variáveis estáticas

```
<?php
  for($i=1; $i<=6; $i++){
    echo display_data($i)."<br>\n";
  }
function display_data($i){
  static $var;
  $var .= "$i";
  return $var;
}
?>
```

variáveis são utilizadas para construir funções que serão capazes de lembrar-se de valores. Especialmente para as recorrências, característica essa muito importante em qualquer sistema. Para demonstrar o que podemos fazer com variáveis estáticas, incluímos um breve exemplo na **Listagem 4**.

Uma repetição garante que *display_data* será chamada seis vezes. A variável *$var* foi definida como uma variável estática, o que significa que lembrará do valor que teve da última vez em que *display_data* foi chamado. Sempre que a função for chamada, o valor atual de *$i* é acrescentado a *$var*. Se executarmos a página, veremos que o tamanho de *$var* cresce sempre que *display_data* é processado conforme podemos ver a seguir:

```
1
12
123
1234
12345
123456
```

As variáveis estáticas têm tanto desvantagens quanto vantagens. Erros de execução provocados por variáveis estáticas são, às vezes, difíceis de encontrar, e pode ser muito trabalhoso procurar erros que só são encontrados por depuração extensiva.

Criação de Packages (Pacotes)

Mesmo escrevendo aplicações bem pequenas, é recomendado empacotar em arquivos separados as funções utilizadas para certas partes das aplicações. Ao escrevermos uma aplicação utilizada para exibir diagramas de estoque, por exemplo, pode ser útil ter bibliotecas separadas de funções relacionadas à recuperação de dados e outras para exibir os dados. Isto nos ajudará a reutilizar o código para outras aplicações. Além do mais, as nossas funções serão mais consistentes, pois nem toda aplicação tem a sua própria implementação.

Escrever código reutilizável é o primeiro passo para obtermos código confiável. Quanto mais freqüentemente um trecho de código é utilizado, mais maduro estará, pois terá sido mais testado. Outro aspecto importante é que os pacotes devem ser independentes dos programas que os usam. Em outras palavras, não devemos acessar variáveis globais do programa principal a partir de uma biblioteca, pois o código poderá não mais ser reutilizável e o programador que utiliza uma biblioteca deve saber o que está acontecendo dentro da mesma.

Devemos tentar construir módulos pequenos, compreensíveis e independentes, com uma interface fixa e garantir que o usuário de uma biblioteca não irá saber o que está acontecendo dentro da mesma.

O fragmento de código da **Listagem 5** mostra um pequeno exemplo. O objetivo é escrever um módulo que troca o conteúdo de duas variáveis. Vejamos antes o programa principal:

No início do *script*, uma biblioteca está sendo inclusa e nela duas variáveis são definidas. Estas são as variáveis que desejamos trocar. Então a função *swapdata_ref* é chamada e os valores são passados para a função. Depois disto, as variáveis são exibidas para que possamos verificar que foram realmente trocadas.

Até agora, o *swapdata_ref* é uma caixa preta, e não será necessário saber como as variáveis são trocadas, porque a interface da função foi claramente definida. Contudo, daremos uma espiada no que a função faz internamente observando a **Listagem 6**.

Uma referência é passada para a função e os valores serão trocados utilizando uma variável temporária chamada *$c*. Como podemos notar a função não tem nenhum retorno. Passada a *swapdata_ref* uma referência ela é capaz de acessar as variáveis diretamente (utilizando os endereços internos dos valores), sem necessidade de acessá-las utilizando variáveis globais.

Este modo de resolver o problema é muito cômodo, porque *swapdata_ref* não precisa saber como é que são chamadas as variáveis na função chamadora. Os endereços dos valores que têm de ser processados serão destinados aos nossos novos nomes quando a função for chamada. Executando o arquivo, podemos ver que as variáveis foram trocadas:

```
a: 37
b: 23
```

Listagem 5. Fragmento de código de biblioteca

```
<?php
  include("lib/transform_data.php");
  $a = 23;
  $b = 37;
  swapdata_ref($a, $b);
  echo "a: $a<br>";
  echo "b: $b";
?>
```

Listagem 6. Código da função swapdata_ref

```
<?php
function swapdata_ref(&$x, &$y){
  $c = $x;
  $x = $y;
  $y = $c;
}
?>
```

Listagem 7. Formulário em HTML

```
<html>
<body>
    Um simples formulário
    <br><br>
    <form action="reaction.php" method="POST">
        <input type="text" name="nome" size="10"><br><br>
        <input type="submit" name="submitbutton">
    </form>
</body>
</html>
```

Listagem 8. Código do script reaction.php

```
<?php
  if($nome){
    echo "nome: $nome";
  }
  else{
    echo "Nada foi digitado no campo.";
  }
?>
```

Como em outras funcionalidades, as referências foram herdadas do C e são uma característica poderosa, pois seria pouco confortável trabalhar com variáveis globais ou passagem de enormes quantidades de dados para uma função.

Trabalhando com formulários

Até agora, vimos como escrever aplicações PHP simples. Contudo, para construir *websites* interativos, é necessário permitir que o usuário digite informações através dos, assim chamados, formulários. O conteúdo de tais formulários pode ser transmitido ao servidor e um *script* PHP pode reagir baseado nos dados enviados a ele. A interação com um servidor *Web* pode ser feita via dois métodos chamados *GET* e *POST*.

Os métodos *GET* e *POST* são descritos no documento *RFC2068* que descreve como devem se comportar. Isso faz parte da regulamentação do protocolo HTTP e pode ser lido no link *www.faqs.org/rfcs/rfc2068.html* que contém mais de 160 páginas. Nesta seção aprenderemos sobre as idéias principais por trás do *GET* e *POST* e aprenderemos também a utilizar esses dois métodos eficientemente. Vejamos o que cada um significa:

- *Get*: O método *GET* é o método de solicitação padrão para recuperar dados de um servidor *Web*. Quando chamamos um *website*, o *GET* é utilizado para obter o documento selecionado. Normalmente, as chamadas deste tipo não têm efeitos colaterais e é por este motivo que *GET* deve ser utilizado; Os navegadores assumem que *GET* não tem nenhum efeito colateral e se a página não mais estiver no *cache* do navegador, será então recuperada novamente. Contudo, se a solicitação original fosse via o *POST*, o usuário receberia uma mensagem alertando que o documento não está mais no *cache*. Mais adiante veremos como trabalhar com ambos métodos;
- Post: O *POST* é o método padrão para submeter dados ao servidor *Web* fornecidod via formulário. No caso do *POST*, a solicitação sempre contém uma seção *body*("corpo") onde a informação relacionada à mesma é fornecida. Esta informação é codificada como uma *query string* ("*string* de consulta").

Normalmente, os desenvolvedores *Web* usam o *POST* mesmo quando nenhum dado no servidor tiver sido modificado;

Criando formulários

Depois que vimos quais métodos são utilizados para recuperar dados de um servidor *Web*, daremos uma olhada em um formulário simples, que pode ser utilizado para enviar o texto a um arquivo PHP. A **Listagem 7** mostra um formulário simples em HTML. Salve o arquivo como "formulario.htm".

Podemos ver que um formulário foi definido na quinta linha através da *Tag* <*form*> do HTML. Quando o usuário clicar no botão *Submit*, os dados serão enviados para o *script reaction.php*. O formulário contém dois componentes. O primeiro componente é o campo de texto chamado *nome*. O segundo componente é o botão para submeter o formulário. O

Listagem 9. Formulário de cadastro

```
<html>
<body>
    Formulário mais complexo
    <br><br>
    <form action="reaction.php" method="POST">
        <input type="text" name="nome" size="10"><br>
        <hr>
         <input type="checkbox" name="box_1">
            display time<br><hr>
         <input type="radio" name="myradio" value=1>Value 1<br>
        <input type="radio" name="myradio" value=2>Value 2<br>
        <input type="radio" name="myradio" value=2>Value 3<br><hr>
         <input type="password" name="passwd">Enter passwords here<br><hr>
         <input type="file" name="filename">enter the filename<br><hr>
         <input type="reset" name="resetbutton">
        <input type="submit" name="submitbutton">
    </form>
</body>
</html>
```

Listagem 10. Código alterado do reaction.php

```
<?php
  echo "nome: $field_1<br>";
  echo "myradio: $myradio<br>";
  echo "passwd: $passwd<br>";
  echo "filename: $filename<br>";
?>
```

Listagem 11. Novo formulário com Seleção

```
<html>
<body>
    Seleção:
    <br><br>
    <form action="reaction.php" method="POST">
        <select name="fruits[]" multiple size=4>
            <option value="Orange">Orange
            <option value="Banana">Banana
            <option value="Apple">Apple
            <option value="Pineapple">Pineapple
            <option value="Strawberry">Strawberry
            <option value="Cherry">Cherry
            <option value="Coconut">Coconut
        </select><br><br>
        <input type="submit" name="submitbutton">
    </form>
</body>
</html>
```

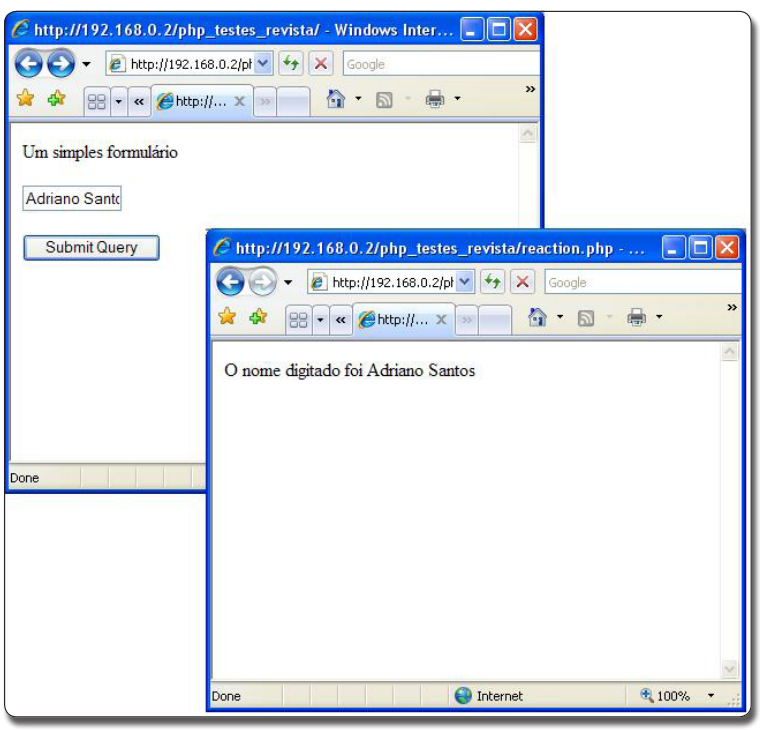


Figura 1. Arquivo de cabeçalho

fim do formulário é marcado pela *Tag </form>*. Crie agora um novo arquivo chamado "reaction.php" e digite o código da **Listagem 8**. Abra o arquivo *formulario.htm* no browser, digite um texto qualquer e clique no botão. Observe pela **Figura 1** que o texto é passado e visto na página *reaction.php*.

A primeira coisa que o *script* faz, é verificar se *$nome* foi definido, ou seja, digitado. Se o usuário tiver inserido dados no formulário, a variável será automaticamente definida pelo PHP. Em contraste com o Perl, os programadores PHP não precisam extrair as variáveis eles próprios a partir da *query string*, todo o trabalho é feito pelo PHP. Se *$nome* foi definido, o conteúdo da variável será exibido na tela.

Como podemos ver, a recuperação de dados de um formulário é uma tarefa fácil. A seguir veremos como formulários mais complexos são construídos com o auxílio do HTML. Modifique o *formulario.htm* de maneira que seu código HTML seja igual a **Listagem 9**. Já o arquivo *reaction.php* deverá se tornar igual ao código da **Listagem 10**. Faça as alterações necessárias e refaça o teste.

O primeiro componente definido no formulário é novamente um campo de texto. O campo tem comprimento 10. Além do tamanho do campo, também seria possível definir o comprimento máximo do texto que se permite que o usuário insira. Isto pode ser feito utilizando o atributo *maxlength*. Depois de definir o campo de texto, um *Checkbox* chamado de *box_1* é definido. Em seguida, podemos ver como os *RadioButtons* são acrescentados a um documento HTML. No exemplo, podemos ver que *myradio* consiste de três componentes. Só um dos três componentes pode ser ativado. Dependendo de qual dos três botões é marcado, o valor apropriado é enviado para *reaction.php*.

Se campos para inserir senhas forem necessários, o componente *Password* será do tipo de componente apropriado. Enquanto estiver digitando, o usuário só verá asteriscos, portanto, ninguém

poderá capturar a senha. Para selecionar um arquivo no disco rígido do usuário, o tipo chamado *file* pode ser utilizado. O componente *file* criará uma caixa de texto e um botão, no qual o usuário irá clicar, caso seja necessária uma janela para selecionar o arquivo usando uma interface gráfica. Finalmente, o botão *Submit* é acrescentado ao formulário. A **Figura 2** mostra o formulário criado pelo código da **Listagem 10**.

O script exibe os valores na tela conforme o que podemos ver em seguida:

```
nome: 23
myradio: 2
passwd: a password
filename: /home/hs/boot.img
```

Repare que os valores na listagem, mostram os dados inseridos no formulário. O importante no exemplo, é que a informação de cada campo no formulário HTML, será armazenada em uma variável após ser submetida a um arquivo PHP.

Às vezes, é necessário dar ao usuário a oportunidade de selecionar mais do que apenas um valor em uma lista. Por isso, o HTML oferece uma solução simples utilizando *select* e *option*. Vamos fazer um novo exemplo**.** Altere novamente o arquivo *formulario.htm* digitando o código da **Listagem 11**. Em seguida altere também o código do arquivo *reaction.php* usando o código a seguir:

```
<?php
  foreach ($fruits as $var){
    echo "$var<br>";
  }
?>
```

Agora o usuário pode escolher algum dos frutos apresentados na lista. Desde que mais de um fruto pode ser selecionado, a estrutura de dados utilizada pelo PHP deve ser um *array*. No nosso exemplo, chamamos este *array* de *fruits[]*. Depois de submeter o conteúdo do formulário, o *array* conterá todos os valores que o usuário selecionou.

Passando parâmetros para um script

Em muitos casos, é interessante chamar um *script* e passar-lhe alguns parâmetros. Técnicas como estas, muitas vezes serão necessárias para os *banners*, pois a pessoa que pagou pelo *banner* quer saber de onde vêm os cliques. Com este objetivo, os parâmetros serão passados para um *script* que contém a informação de qual *website* contém o *banner*. Vejamos um *script* muito simples, que não faz nada exceto exibir duas variáveis:

```
<?php
  echo "a: $a<br>\n";
  echo "b: $b\n";
?>
```

O objetivo é passar parâmetros para o script via uma URL como por exemplo:

```
http://localhost/test/script.php?a=234&b=197
```

O *script* chamado *script.php*, localizado em um diretório denominado *test* no servidor *Web* local, é chamado com o parâmetro *a=237* e *b=197*. Os nomes do *script* e os parâmetros são separados utilizando um caractere "?" (sinal de interrogação). A lista de parâmetros é separada utilizando o caractere "@" (arroba). Executando *script.php*, o resultado será exibido pelo navegador:

```
a: 234
b: 197
```

Nós já vimos que as marcas de sinal de interrogação e arroba são utilizados como delimitadores. O que acontece se desejamos passar um desses sinais para o *script*? O servidor *Web* irá confundir-se, pois terá que descobrir os símbolos que são utilizados como delimitadores daqueles que são partes de uma string. A recomendação é: todos os caracteres "especiais" têm que ser evitados para que o servidor *Web* possa analisar a *URL* facilmente.

O PHP provê uma função chamada *urlencode*, que toma conta deste tipo de situação. Vamos escrever um pequeno *script* que gera uma lista de *URLs*: Portanto crie uma nova página e nela digite o código da **Listagem 12**.

Depois de exibir o código HTML, cinco *strings* serão definidas e atribuídas a um array chamado *$messages*. Cada uma das *strings* contém certos caracteres que têm que ser evitados pelo PHP. No passo a seguir, as conexões serão geradas utilizando o *urlencode*. Vamos chamar o PHP via linha de comando e verificar o aspecto do código HTML gerado pelo *script*. Digite na barra de endereços uma das *urls* a seguir:

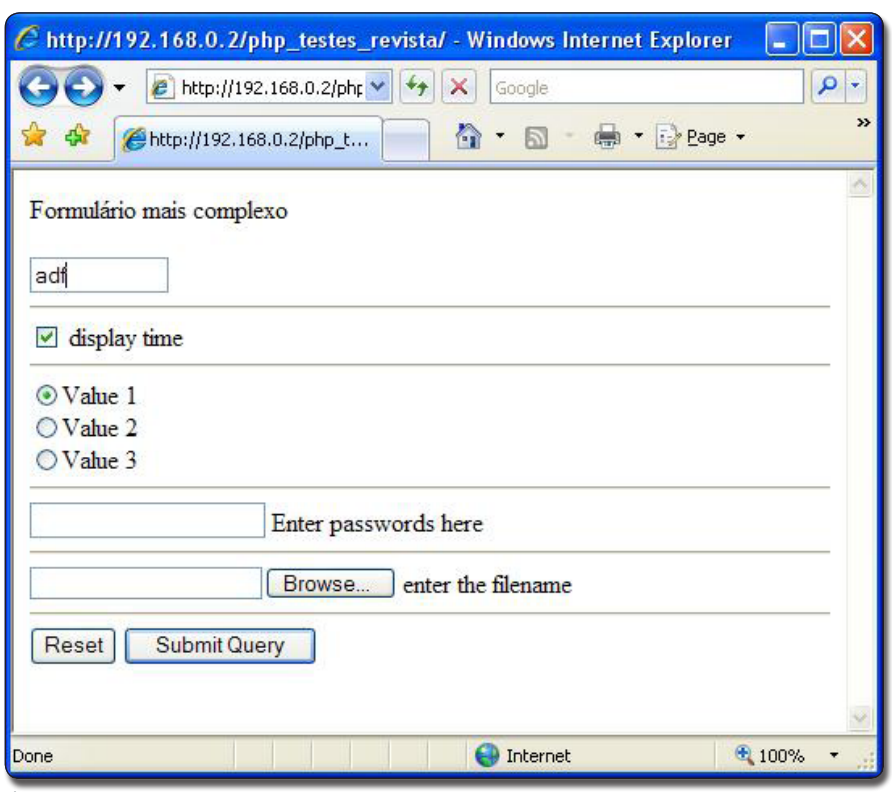


Figura 2. Formulário um pouco mais complexo

```
http://localhost/reaction.
php?a=Pat+and+Shelley
http://localhost/reaction.
php?a=Pat+%26+Shelley
http://localhost/reaction.
php?a=Pat+%26+Shelley%21
http://localhost/reaction.
php?a=Are+you+sure%3F
http://localhost/reaction.php?a=Hans-
J%FCrgen+Sch%F6nig
```

Como podemos ver, todos os caracteres especiais foram evitados. Executando o *script* usando um navegador *Web*, as *strings* no *array* serão exibidas como links:

```
Pat and Shelley
Pat & Shelley
Pat & Shelley!
Are you sure?
Hans-Jürgen Schönig
```

Se clicarmos no primeiro link, o *script.php* será chamado:

```
a: Pat and Shelley
b:
```

O texto é exibido na tela. Como não foi passado ao *script* nenhum parâmetro para *$b*, o campo estará vazio.

Trabalho com campos ocultos

Às vezes, é necessário passar parâmetros para um *script* que não devem ser vistos pelo usuário. No HTML, é possível utilizar campos ocultos. A diferença entre campos "normais" e campos ocultos é que os campos ocultos não serão exibidos pelo navegador. Por isso, os parâmetros são facilmente passados de um *script* para o outro sem que o usuário consiga modificar a informação contida em um campo oculto. Supondo que desejamos escrever um *script* que exibe a data e hora em que o *script* anterior foi criado (**Listagem 13**)

Listagem 12. Usando o urlencode

```
<?php
  echo "<html><body>\n";

  $messages[0] = "Pat and Shelley";
  $messages[1] = "Pat & Shelley";
  $messages[2] = "Pat & Shelley!";
  $messages[3] = "Are you sure?";
  $messages[4] = "Hans-Jürgen Schönig";

  foreach ($messages as $var){
    echo '<a href="script.php?a='.urlencode($var).
      '"> '.$var."</a><br>\n";
  }
  echo '</body></html>';
?>
```

Listagem 13. Script criado com campo hidden, oculto

```
<?php
  $curtime = localtime();
  $timestr = strftime("%Y %b %d: %T %Z");
  echo "timestr: $timestr<br>\n";
  echo '
    <html>
      <body>
        Olá mundo!
        <form action="reaction.php" method="POST">
          <input type="hidden" value="'.$timestr.
            '" name="gentime">
          <input type="submit" name="submitbutton">
        </form>
      </body>
    </html>
  ';
?>
```

O primeiro *script* gera a data e hora atuais e os armazena em *$timestr*. Agora, um formulário é gerado e o valor de *$timestr* é utilizado como o valor *default* do chamado *gentime*. As únicas informações exibidas na tela serão: o conteúdo de *$timestr*, a string "Olá Mundo!" e um botão para submeter à informação. O *reaction.php* é chamado quando o usuário clica no botão. Vejamos o *reaction.php*:

```
<?php
  echo "Hora gerada: $gentime";
?>
```

O conteúdo de *$gentime* é exibido na tela:

```
gentime: 2007 Nov 01: 12:31:57 CET
```

O script anterior, foi gerado na data e hora listadas na saída de *reaction.php*. Às vezes, é interessante passar a data e hora de geração do primeiro HTML para todos os arquivos, pois deste modo, é possível descobrir quando um usuário acessou a página.

Conclusão

O PHP é uma linguagem poderosa e fácil de se trabalhar. As vantagens principais do PHP são a sua flexibilidade e a sua simplicidade. As aplicações Web podem ser escritas fácil e rapidamente. Isto faz com que o PHP se torne uma boa escolha para a rápida prototipação e o desenvolvimento de aplicações.

Veremos muitos outros artigos sobre a linguagem PHP, porém é de suma importância que se estude cada vez mais o script, procurando entender melhor como tudo funciona. Procure desenvolver aplicações simples com formulários, transferência de valores entre as páginas, uso de sessões e uma série de outras funcionalidades disponíveis no PHP. Bons códigos e até a próxima. ●

Seu potencial. Nossa inspiração.™
Microsoft
ROMA LEVOU MIL ANOS PARA SER CONSTRUÍDA.
SUA EQUIPE TEM UM MÊS.
ENFRENTE OS DESAFIOS
Desafio: terminar projetos astronômicos na velocidade da luz. Estratégia: mais comunicação e colaboração, com o Visual Studio® Team System. Veja dicas e ferramentas em defyallchallenges.com
Microsoft Visual Studio
©2007 Microsoft Corporation. Todos os direitos reservados Microsoft, Visual Studio, o logo do Visual Studio e "Seu potencial. Nossa inspiração", são marcas comerciais, registradas ou não, da Microsoft Corporation nos Estados Unidos e/ou em outros países. Os nomes das companhias ou produtos reais aqui mencionados podem ser marcas comerciais de seus respectivos proprietários.

NOVOS PLANOS

Descubra porque Hospedix é a hospedagem preferida dos profissionais.

Plano **Profissional** 1

Windows Server 2003

1 Registro de domínio grátis*
3 Domínios por conta
50 GB de transferência
30 GB para e-mails com SSL
1 GB de espaço
ASP, ASP.NET 2.0 e PHP 5
Access e MySQL 5
SQL Server 2005 Express
Painel de controle e webmail
Editor de páginas on-line
Anti-spam e Anti-vírus

R$ 25,90 por mês

30 DIAS GRÁTIS
PARA EXPERIMENTAR
Ao assinar, digite o código de desconto abaixo exclusivo para leitores da .NET Magazine e ganhe o primeiro mês inteiramente grátis.

RVSTNET

DOMÍNIO GRÁTIS
ENQUANTO FOR CLIENTE
Só na Hospedix você tem registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

PAINEL DE CONTROLE
PLESK 8.1 PROFISSIONAL
Painel de controle completo onde você mesmo adiciona domínios, sub-domínios, cria contas de e-mail, e muito mais.

Transfira seu site hoje mesmo para a Hospedix e descubra porque somos a empresa de hospedagem preferida dos profissionais.

www.hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL